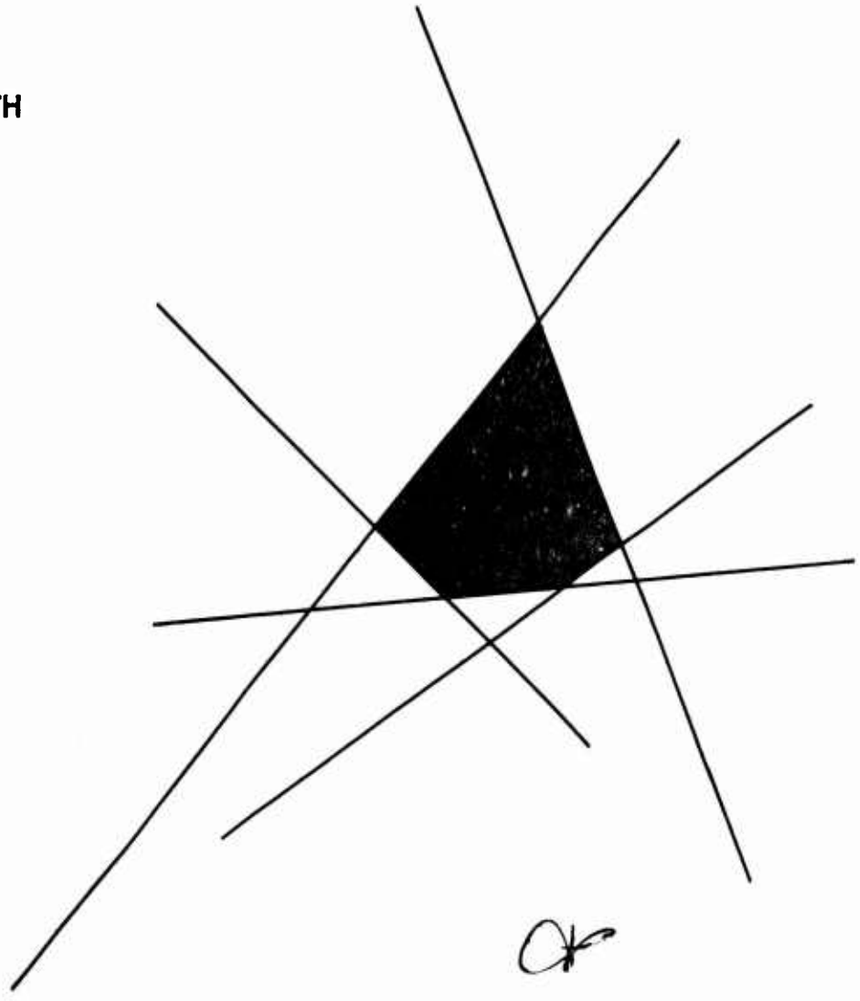
ORC 69-21
AUGUST 1969

A BRANCH AND BOUND METHOD FOR OPTIMAL FAULT FINDING

by
RICHARD W. BUTTERWORTH



OPERATIONS
RESEARCH
CENTER

COLLEGE OF ENGINEERING
UNIVERSITY OF CALIFORNIA · BERKELEY

A BRANCH-AND-BOUND METHOD FOR FAULT FINDING

by

Richard W. Butterworth
Operations Research Center
University of California, Berkeley

AUGUST 1969 ORC 69-21

This research has been partially supported by the U. S. Army Research Office-Durham under Contract DA-31-124-ARO-D-331 and the National Science Foundation under Grant GK-1684 with the University of California. Reproduction in whole or in part is permitted for any purpose of the United States Government.

ACKNOWLEDGEMENT

The author wishes to thank Professors E. Lawler and R. Wolff for their interest in and guidance of this writing. In addition, the author's wife deserves thanks for her clerical and moral support.

ABSTRACT

The problem is that of optimally testing a coherent system to learn some characteristic of it, for example, whether it is operating or not. A branch and bound and a dynamic programming solution are given, as well as a comparison of computer computation times for both. Several specific models with analytical solutions are also presented.

The general problem is posed abstractly in Chapter 1, and two solution methods are detailed. Briefly, we are presented with n binary-valued random variables and a function of the vector of these random variables. The object is to learn the value of the function by testing some of the random variables and an optimal testing policy uses the minimum average time to complete its testing. The first solution method given is a dynamic programming type as this is easier to formulate. The second is a branch and bound method; its description is somewhat more involved. Chapter 2 is concerned with formulating general fault finding models which can be cast as problems solved by the methods presented in Chapter 1. Also included is some computational experience comparing the effectiveness of both solution methods for one of the models described. Chapter 3 gives several specific fault finding problems and analytical results for them. This chapter is developed independently of Chapter 1, and relies on Chapter 2 only for its initial discussion of coherent systems.

TABLE OF CONTENTS

CHAPTER 1

DECISION PROBLEMS

Introduction

Decision problems occur when one is presented with the task of determining the value of a known function of n random variables, in the following circumstance. We suppose that the n random variables are binary, each one taking the value zero or one. There is a time $t_i \geq 0$ incurred to determine the value of the ith random variable. We remark that, of course, t_i could be interpreted as any other measure of the cost to determine the value of the ith random variable. A procedure, or policy, for determining the value of the known function is a rule which tells its user which random variable to test next, based on the results of the previously made tests, or to stop, if the known function can have only one value, almost surely, given the results of tests made up to this point.

When the function is one to one, determining its value is equivalent to determining the value of every random variable. In general, however, the function is many to one. The collection of inverse images of each functional value forms a partition of the sample space; determining the function's value is then equivalent to identifying the set of the partition in which the vector of random variables resides.

There are various criteria for comparing policies; we will deal primarily with the expected time until the testing process stops and the value of the known function has been determined almost surely. A straightforward dynamic programming solution and a branch and bound type solution are presented, as well as an example illustrating the branch and bound algorithm. The chapter is concluded with a brief discussion of another criterion.

The generality of this model of decision problems suggests that a wide variety of real problems can be formulated in these terms. One such problem is that of converting "decision tables", a format for computer programming, into a set of machine instructions logically equivalent to the given decision table. See, for example, [Reinwald and Soland, 1966] and [Reinwald and Soland, 1967]. Some fault finding problems can also be formulated this way, which is the topic of Chapter 2. A frequent suggestion as a real application is clinical diagnosis.

A. Definitions and Notation

We say $(Y_1, \ldots, Y_n ; A_1, \ldots, A_d)$ is a n-th order decision problem when:

(i) $Y_1, \ldots, Y_n$ are binary (0 or 1) random variables

and

(ii) $\{A_1, \ldots, A_d\}$ is a partition of $\{0, 1\}^n = \{(\omega_1, \ldots, \omega_n) \mid \omega_i = 0 \text{ or } 1 , i = 1, \ldots, n\}$.

The correct action is k , a random variable, defined by $(Y_1, \ldots, Y_n) \in A_k$ almost surely. We put $Y = (Y_1, \ldots, Y_n)$ and write $Y \in A_k$ a.s. The object is to determine the correct action almost surely in minimum expected time by successively determining the values of the Y_j's. The "correct action" should be thought of as the value of the function mentioned in the introduction and should not be confused with a policy for determining it. This terminology is motivated by applications, such as in Chapter 2.

To determine, or test, Y_j requires time $t_j \geq 0$. The process stops when we are almost sure of the value of random variable k . We say the state of the problem is $s = (s_1, \ldots, s_n) \in \{-1, 0, 1\}^n$ when, if $s_j = -1$ then Y_j has not been tested, while if $s_j = 0$ or 1 then

Y_j has been tested and its value is s_j. In particular, set $s^o = (-1, \ldots, -1)$, the initial state when no Y_j has been tested. We define $\hat{s} = \{Y_j = s_j \mid j \ni s_j \neq -1\}$, the event in the underlying probability space corresponding to the state of knowledge s.

A policy for $(Y_1, \ldots, Y_n ; A_1, \ldots, A_d)$ is a rule which tells its user which Y_j to test next, based on the state of the problem. Specifically, to use policy π when in state s means to test $Y_{\pi(s)}$, if $\pi(s) = 1, 2, \ldots, n$, and to stop if $\pi(s) = 0$, indicating the correct action is almost surely determined. We find it formally convenient to require that policies stop at states s for which $P(\hat{s}) = 0$, and at those states when the correct action is known, almost surely. Accordingly, <u>a policy</u> will be any function π on $\{-1, 0, 1\}^n$ to $\{0, 1, \ldots, n\}$ such that for states $s = (s_1, \ldots, s_n)$.

(1) $P(\hat{s}) = 0 \Rightarrow \pi(s) = 0$

(2) $P(\hat{s}) > 0 \Rightarrow$
- (a) $\pi(s) = 0 \iff \exists k \ni P\{Y \in A_k \mid \hat{s}\} = 1$ and
- (b) $\pi(s) = j > 0 \Rightarrow s_j = -1$.

We will denote by $T_\pi(s)$ the (conditional) expected time to determine the correct action, beginning in state s and following policy π. Specifically,

$T_\pi(s)$ = expected time until process stops, beginning in state s and using policy π, with respect to the conditional distribution $P\{\cdot \mid \hat{s}\}$,
= 0 if $P(\hat{s}) = 0$.

There are, of course, various criteria for comparing policies. Our interest here is in the expected time to determine the correct action, beginning in the initial state s^o. Accordingly, we will call policy

π_o optimal if it minimizes $T_\pi(s^o)$ among all policies.

B. Dynamic Programming Solution

Let us enrich the problem by assuming the time to test Y_j depends on the state s of the problem when the test is performed, say $t(j, s)$. This being the case, we have for all states $s \ni P(\hat{s}) > 0$, for all policies $\pi \ni \pi(s) = j > 0$, $T_\pi(s) = t(j, s) + P\{Y_j = 0 \mid \hat{s}\}T_\pi(s') + P\{Y_j = 1 \mid \hat{s}\}T_\pi(s'')$, where $s' = (s_1, \ldots, 0_i, \ldots, s_n)$ and $s'' = (s_1, \ldots, 1_i, \ldots, s_n)$. This provides the basis for the usual dynamic programming solution to finding an optimal policy. Let the optimal value function be $T(s) = \text{Min}\{T_\pi(s) \mid \text{policies } \pi\}$. Then

$$T(s) = \begin{cases} 0 \text{ if } \exists \text{ policy } \pi \ni \pi(s) = 0, \text{ i.e., which stops at } s; \\ \text{otherwise} \\ \quad \text{Min} \quad (t(j, s) + P\{Y_j = 0 \mid \hat{s}\}T(s') + P\{Y_j = 1 \mid \hat{s}\}T(s'')) \\ j \ni s_j = -1 \\ \text{where } s' = (s_1, \ldots, 0_j, \ldots, s_n) \text{ and } s'' = (s_1, \ldots, 1_j, \ldots, s_n). \end{cases}$$

This functional equation can be solved by considering states s with successively more occurrences of -1. An optimal policy is also generated in the usual way.

The primary restriction to using this solution is that the functional equation must be used once for each state, indicating that the computational complexity of the algorithm grows roughly as 3^n. On the other hand, by making these computations we have really solved a family of related problems, each one an extension of the original. This bonus is a common occurrence in the solution of problems via dynamic programming. The extension is simply the assumption that the cost of determining Y_j can be allowed to depend on the tests previously made and their results, namely the current state of the problem. Also, we see that an optimal

policy π_o recovered from the functional equation above really satisfies: $\forall s : T_{\pi_o}(s) = \min_{\pi} T_{\pi}(s)$. That is, π_o is optimal uniformly for beginning in any state s . This is stronger than simply requiring it minimize $T_{\pi}(s^o)$. If, during the use of this optimal policy, some more information were to become available, then the problem need not be resolved for a new initial state; one can continue to use the optimal policy generated by the functional equation, as it will be optimal for this newly created problem.

C. Branch and Bound Solution

The decision problem has a "branch and bound" solution which, like any solution of this type, is potentially an enumeration of all feasible solutions. Such complexity is usually not realized and this solution provides a reasonable alternative to the dynamic programming solution.

In order to describe the branch and bound scheme precisely, the notion of a policy tree will be defined. Roughly, a policy tree is a partial specification of a testing policy. For example, the policy tree $\{(s^o, i)\}$ is the partial specification $\pi(s^o) = i$, so we say this policy tree represents all policies which begin by testing Y_i first (in state s^o). Thus, policy trees become the device for "branching" at each step. The "bounding" of policy trees, that is, a lower bound on $T_{\pi}(s^o)$ for all π represented by some policy tree, is obtained from a lower bound on the probability that each Y_j will be tested. An appealing feature of the bound obtained occurs when the algorithm branches on a particular policy tree by making its specification more complete in various ways and hence creating a collection of related trees; the lower bounds for the newly created trees are computed by simply adding a quantity to the lower bound for the former tree being expanded. These incremental

quantities also provide an upper bound on $T_{\pi_o}(s^o)$ for an optimal policy π_o. This upper bound is used to eliminate some policy trees from consideration directly. While shuch trees would never be expanded in the normal course of compu:ation anyway, their very presence consumes storage and time in needless manipulation.

The basic form of our algorithm is the content of [Soland 1966], except for the probabilistic interpretation of the lower bounds and the computation and use of upper bounds.

A policy tree will be a partial specification for policies, and will be the device which allows us to use a branch and bound scheme. Because of the inductive way this device will be used, we will define a policy tree inductively. A <u>policy tree</u> is a set $\mathcal{Q}$ of ordered pairs (s, i) of states s and integers i, $0 \le i \le n$, such that:

<u>Basis</u>: $\mathcal{Q} = \{(s^o, i)\}$ is a policy tree, provided there exists a policy π such that $\pi(s^o) = i$.

<u>Induction</u>: If $\mathcal{Q}$ is a policy tree and $(s, i) \in \mathcal{Q}$ such that $i > 0$ and, for $s' = (s_1, \ldots, 0_i, \ldots, s_n)$ and $s'' = (s_1, \ldots, 1_i, \ldots, s_n)$, neither (s', j) nor (s'', k) are elements of $\mathcal{Q}$ for any j and k, then $\mathcal{Q} \cup \{(s', h'), (s'', h'')\}$ is a policy tree for all pairs h', h'' such that there exists a policy π with $\pi(s') = h'$ and $\pi(s'') = h''$.

We need to define several quantities related to policy trees. First, the policy tree $\mathcal{Q}$ will be said to <u>represent the policy</u> π when, for all $(s, i) \in \mathcal{Q}$, we have $\pi(s) = i$. In this way, policy trees partially specify a policy. Notice that the larger the policy tree is, the fewer policies it represents. When a policy tree is as large as possible, that

is, when it cannot be extended according to the induction step of the definition above, we say it is a complete policy tree. Equivalently, a policy tree is complete if it is set maximal among all policy trees. We will see below that complete policy trees effectively represent only one policy. One final notion is that of a terminal element of a policy tree. The element $(s, i) \in \mathcal{A}$ is a terminal element of $\mathcal{A}$ if $i = 0$ or, if $i > 0$, then neither (s', j) nor (s'', k) are elements of $\mathcal{A}$ for any j and k, where $s' = (s_1, \ldots, 0_i, \ldots, s_n)$ and $s'' = (s_1, \ldots, 1_i, \ldots, s_n)$. Elements of $\mathcal{A}$ which are not terminal are called interior. We see the terminal elements (s, i) of a policy tree for which $i > 0$ are exactly those referred to in the induction step of the policy tree definition.

We continue with some basic properties of policy trees.

Lemma 1: Every policy tree represents at least one policy.

Lemma 2: A policy tree is complete <=> for all terminal elements (s, i) of $\mathcal{A}$, we have $i = 0$ => all policies represented by $\mathcal{A}$ are equivalent to the user, i.e., implementation of any two policies represented by $\mathcal{A}$ a.s. lead to the same sequence of tests.

Lemma 3: Let $\mathcal{A}$ be any policy tree and π be any policy represented by $\mathcal{A}$. Then

$$(1) \qquad T_\pi(s^0) = \sum_{\substack{(s,\,i) \text{ interior} \\ \text{elements of } \mathcal{A}}} P(\hat{s})\, t_i + \sum_{\substack{(s,\,i) \text{ terminal} \\ \text{elements of } \mathcal{A}}} P(\hat{s})\, T_\pi(s) .$$

Proofs: Lemmas 1 and 2 are easily shown by induction. Lemma 3, also shown by induction, is proven here. We will show that Equation (1) holds for all policies represented by $\mathcal{A}$ by showing inductively that the class of policy trees for which (1) holds contains all policy trees.

Basis: If $\mathcal{A} = \{(s^o, i)\}$, then (1) clearly holds; (s^o, i) is a terminal element.

Induction: Let $\mathcal{A}$ be any policy tree for which (1) holds. Let (s, i) be a terminal element of $\mathcal{A}$ with $i > 0$. Let $s' = (s_1, \ldots, 0_i, \ldots, s_n)$ and $s'' = (s_1, \ldots, 1_i, \ldots, s_n)$. If (h', h'') are such that for some policy π , $\pi(s') = h'$ and $\pi(s'') = h''$, then $\mathcal{B} = \mathcal{A} \cup \{(s', h'), (s'', h'')\}$ is a policy tree also. We show (1) holds for $\mathcal{B}$. If π is represented by $\mathcal{B}$, then π is represented by $\mathcal{A}$, $\pi(s') = h'$ and $\pi(s'') = h''$. Hence

$$P(\hat{s})T_\pi(s) = P(\hat{s})\left[t_i + P\{Y_i = 0 \mid \hat{s}\}T_\pi(s') + P\{Y_i = 1 \mid \hat{s}\}T_\pi(s'')\right]$$

$$= P(\hat{s})t_i + P(\hat{s}')T_\pi(s') + P(\hat{s}'')T_\pi(s'') .$$

It now follows (1) holds for $\mathcal{B}$. (s', h') and (s'', h'') are terminal elements of $\mathcal{B}$, while (s, i) becomes an interior element of $\mathcal{B}$. //

A lower bound on $T_\pi(s^o)$ over policies represented by a specific policy tree is generated by Lemma 3 and the following probabilistic considerations. Let π_j be any policy satisfying: $\forall s$, $\pi_j(s) = j \Rightarrow \forall i \neq j$, $s_i \neq -1$. Thus π_j tests Y_j only after all other Y_i have been tested. Many policies have this property, but the random event of testing Y_j is the same for each such policy. Letting π be any policy, we have, for the problem beginning in the nominal initial state s^o ,

(2a) $\{Y_j$ is tested by using policy $\pi_j\} \subseteq \{Y_j$ is tested by using policy $\pi\}$.

It follows that, for any state $s \ni P(\hat{s}) > 0$.

(2b) $P\{\text{beginning in state } s,\ Y_j \text{ is tested by using policy } \pi_j \mid \hat{s}\} \leq P\{\text{beginning in state } s,\ Y_j \text{ is tested by using policy } \pi \mid \hat{s}\}$.

Lemma 4: Let π_j be as described above. Let

$$I_j(s) = \begin{cases} t_j P\{\text{beginning in state } s,\ Y_j \text{ is tested by using policy } \pi_j \mid \hat{s}\} \\ \qquad \text{if } P(\hat{s}) > 0 \\ 0 \quad \text{otherwise,} \end{cases}$$

and let $t_o = 0$. Then, for any state s and policy tree $\mathcal{A}$,

(i) for all policies $\pi \ni \pi(s) = i$, $\sum_{\substack{j=1 \\ j\neq i}}^{n} I_j(s) + t_i \leq T_\pi(s)$

(ii) for all policies π represented by $\mathcal{A}$,

$$(3)\quad T_\pi(s^o) \geq \sum_{(s,i)\in\mathcal{A}} P(\hat{s}) t_i + \sum_{\substack{(s,i) \text{ terminal} \\ \text{elements of } \mathcal{A}}} P(\hat{s}) \sum_{\substack{j=1 \\ j\neq i}}^{n} I_j(s) .$$

Denoting this lower bound by $L(\mathcal{A})$, we have

(iii) for all policies π represented by $\mathcal{A}$, $T_\pi(s^o) \geq L(\mathcal{A})$ and, when $\mathcal{A}$ is complete, $T_\pi(s^o) = L(\mathcal{A})$.

Proof: (ii) and (iii) follow from (i) and Lemma 3. (i) follows from Equation (2) above and the formula:

$$T_\pi(s) = \begin{cases} \sum_{j=1}^{n} t_j P\{Y_j \text{ is tested when beginning in state } s \text{ and using } \pi \mid \hat{s}\} \\ \qquad \text{if } P(\hat{s}) > 0 \\ 0 \qquad \text{if } P(\hat{s}) = 0 \text{ . // } \end{cases}$$

An upper bound on the value of an optimal policy can be computed in terms of quantities which will be on hand from making the lower bound computations. Specifically, let $\mathcal{A}$ be any policy tree and π any

policy such that for all interior elements (s, i) of $\mathcal{A}$, we have $\pi(s) = i$. Thus π may not be represented by $\mathcal{A}$. However, Equation (1) remains valid and is proven the same way. Recalling this formula,

$$T_\pi(s^o) = \sum_{\substack{(s,\,i) \text{ interior} \\ \text{elements of } \mathcal{A}}} P(\hat{s})t_i + \sum_{\substack{(s,\,i) \text{ terminal} \\ \text{elements of } \mathcal{A}}} P(\hat{s})T_\pi(s) .$$

Now, let (s, i) be a terminal element of $\mathcal{A}$ for which $i > 0$. Suppose π is such that, beginning in state s, Y_i is not tested unless it is necessary, that is, unless π has tested all untested Y_j (those with $s_j = -1$) except Y_i, and still it is necessary to know Y_i in order to determine the correct action. Thus π is a π_i policy as described above. Then $I_i(s) = t_i P\{Y_i$ is tested, beginning in state s and using policy $\pi \mid \hat{s}\}$ so

$$T_\pi(s) \le \sum_{\substack{j \ni s_j = -1 \\ j \neq i}} t_j + I_i(s) .$$

It follows that for some policy π,

$$(4)\quad T_\pi(s^o) \le \sum_{\substack{(s,i) \text{ interior} \\ \text{element of } \mathcal{A}}} P(\hat{s})t_i + \sum_{\substack{(s,i) \text{ terminal} \\ \text{element of } \mathcal{A} \ni \\ i > 0}} P(\hat{s}) \sum_{\substack{j \ni s_j = -1 \\ j \neq i}} t_j + I_i(s) .$$

Denoting this upper bound by $U(\mathcal{A})$, we have for all policy trees $\mathcal{A}$, $T_{\pi_o}(s^o) \le U(\mathcal{A})$, where π_o denotes an optimal policy, and $T_{\pi_o}(s^o) = U(\mathcal{A})$ whenever $\mathcal{A}$ is complete. Our branch and bound scheme will generate various policy trees and the quantities necessary to compute $U(\mathcal{A})$.

The following formulas give an explicit form for $I_j(s)$ and also allow us to see how the bounds $L(\mathcal{A})$ and $U(\mathcal{A})$ are computed sequentially as needed. Define

$$\Omega(s) = \left\{\omega = (\omega_1, \ldots, \omega_n) \in \{0, 1\}^n \mid \forall j \ni s_j = 0 \text{ or } 1 \text{ , we have } \omega_j = s_j\right\},$$

so in particular, $\Omega = \Omega(s^o) = \{0, 1\}^n$. One can think of $\omega \in \Omega$ as an outcome of the random vector $Y = (Y_1, \ldots, Y_n)$. The probability of such an outcome is $P\{Y = \omega\}$. Let

$$\Delta_j(\omega) = \begin{cases} 1 & \text{if, for } \omega' = (\omega_1, \ldots, (1 - \omega_j), \ldots, \omega_n) \text{ , we have} \\ & P\{Y = \omega\} > 0 \text{ , } P\{Y = \omega'\} > 0 \text{ and } \omega \text{ and } \omega' \text{ belong to} \\ & \text{distinct sets of the partition } \{A_1, \ldots, A_d\} . \\ 0 & \text{otherwise.} \end{cases}$$

$$\bar{\Delta}_j(\omega) = \begin{cases} 1 & \text{if, for } \omega' = (\omega_1, \ldots, (1 - \omega_j), \ldots, \omega_n) \text{ , we have} \\ & P\{Y = \omega\} = 0 \text{ or } P\{Y = \omega'\} = 0 \text{ or } \omega \text{ and } \omega' \text{ belong to} \\ & \text{a common set of the partition } \{A_1, \ldots, A_d\} . \\ 0 & \text{otherwise.} \end{cases}$$

These are devices for computing $I_j(s)$. For states s with $P(\hat{s}) > 0$,

$$I_j(s) = \begin{cases} t_j \sum_{\omega \in \Omega} P\{Y = \omega \mid \hat{s}\}\Delta_j(\omega) & \text{if } s_j = -1 \\ 0 & \text{if } s_j = 0 \text{ or } 1 . \end{cases}$$

However, what really appears in computing $L(\mathcal{A})$ and $U(\mathcal{A})$ is $P(\hat{s})I_j(s)$. For any state s and integer j , $1 \le j \le n$, define

$$E_j(s) = \begin{cases} t_j \sum_{\omega \in \Omega(s)} P\{Y = \omega\}\Delta_j(\omega) & \text{if } s_j = -1 \\ 0 & \text{if } s_j = 0 \text{ or } 1 , \end{cases}$$

so

$$P(\hat{s})I_j(s) = E_j(s) . \tag{5}$$

For any state s and integer j, $1 \le j \le n$, define

$$\bar{E}_j(s) = \begin{cases} t_j \sum_{\omega \in \Omega(s)} P\{Y = \omega\} \bar{\Delta}_j(\omega) & \text{if } s_j = -1 \\ 0 & \text{if } s_j = 0 \text{ or } 1 . \end{cases}$$

Notice $\Delta_j(\omega) + \bar{\Delta}_j(\omega) = 1$, so

$$\bar{E}_j(s) + E_j(s) = t_j P(\hat{s}) . \tag{6}$$

Now, recalling the formulas (3) and (4), we can write $L(\mathcal{A})$ and $U(\mathcal{A})$ in terms of $E_j(s)$.

$$L(\mathcal{A}) = \sum_{(s,\, i) \in \mathcal{A}} P(\hat{s}) t_i + \sum_{\substack{(s,\, i) \text{ terminal} \\ \text{element of } \mathcal{A}}} \sum_{\substack{j=1 \\ j \neq i}}^{n} E_j(s) . \tag{7}$$

$$U(\mathcal{A}) = \sum_{\substack{(s,\, i) \text{ interior} \\ \text{element of } \mathcal{A}}} P(\hat{s}) t_i + \sum_{\substack{(s,\, i) \text{ terminal} \\ \text{element of } \mathcal{A} \ni \\ i > 0}} \left[P(\hat{s}) \sum_{\substack{j \ni s_j = -1 \\ j \neq i}} t_j + E_i(s) \right] . \tag{8}$$

In particular, for a policy tree of the form $\mathcal{A} = \{(s^0, i)\}$ with $i > 0$, using (6) we see

$$L(\mathcal{A}) = \sum_{j=1}^{n} t_j - \sum_{j=1}^{n} \bar{E}_j(s^0) + \bar{E}_i(s^0) . \tag{9}$$

$$U(\mathcal{A}) = \sum_{j=1}^{n} t_j - \bar{E}_j(s^0) . \tag{10}$$

Now, if we extend a policy tree $\mathcal{A}$, we want to obtain the bounds for the new tree in terms of the bounds on $\mathcal{A}$. Specifically, let (s, i') be a terminal element of the tree $\mathcal{A}$ with $i > 0$. For $s' = (s_1, \ldots, 0_i, \ldots, s_n)$ and $s'' = (s_1, \ldots, 1_i, \ldots, s_n)$, if h' and h'' are such that there exists a policy with $\pi(s') = h'$ and $\pi(s'') = h''$, then according to the induction step of the definition of policy tree, $\mathcal{B} = \mathcal{A} \cup \{(s', h')$,

$(s'', h'')\}$ is also a policy tree; using (7), (8) and (6), we have

$$L(\mathcal{B}) = L(\mathcal{A}) + \bar{E}_{h'}(s') + \bar{E}_{h''}(s'') \tag{11}$$

and

$$U(\mathcal{B}) = U(\mathcal{A}) + \bar{E}_i(s) - \bar{E}_{h'}(s') - \bar{E}_{h''}(s'') \tag{12}$$

$$- 1(h' = 0)P(\hat{s}') \sum_{j \ni s'_j = -1} t_j - 1(h'' = 0)P(\hat{s}'') \sum_{j \ni s''_j = -1} t_j$$

where we have taken $\bar{E}_o(s) = 0$ for every state s and

$$1(h = 0) = \begin{cases} 1 & \text{if } h = 0 \\ & , h = h' , h'' . \\ 0 & \text{if } h \neq 0 \end{cases}$$

Let $(Y_1, \ldots, Y_n ; A_1, \ldots, A_d)$ be a decision problem. The following is a "branch and bound" algorithm for finding an optimal policy.

Step 1: If every policy π stops at initial state s^o (corresponding to the correct action being a constant almost surely), then $T_\pi(s^o) = 0$. If not, then $\forall i \ni 1 \leq i \leq n$, $\exists$ policy $\pi \ni \pi(s^o) = i$. Let $\mathcal{F} = \{\mathcal{A}_i \mid i = 1, 2, \ldots, n\}$ where $\mathcal{A}_i = \{(s^o, i)\}$, a policy tree. Notice every policy is represented by exactly one of the policy trees in the family $\mathcal{F}$. Set

$$L(\mathcal{A}_i) = t_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} E_j(s^o)$$

and

$$U(\mathcal{A}_i) = E_i(s^o) + \sum_{\substack{j=1 \\ j \neq i}}^{n} t_j$$

and

$$U = \operatorname{Min}_i U(\mathcal{A}_i) .$$

Then $T_{\pi_o}(s^o) \leq U$ and, for all i and π with $\pi(s^o) = i$, $L(\mathcal{Q}_i) \leq T_\pi(s^o)$. Continue to Step 2.

Step 2: Reduce the set $\mathcal{T}$ by removing any $\mathcal{Q} \in \mathcal{T} \ni L(\mathcal{Q}) > U$; those policy trees cannot represent an optimal policy. Let the policy tree $\mathcal{Q}$ satisfy $L(\mathcal{Q}) = \text{Min } L(\mathcal{Q}')$ over $\mathcal{Q}' \in \mathcal{T}$. If $\mathcal{Q}$ is a complete policy tree (cannot be augmented), then all policies it represents are equivalent to the user (Lemma 2) and any one of them is optimal, since for such a policy π, $L(\mathcal{Q}) = T_\pi(s^o)$. If $\mathcal{Q}$ is not complete, continue to Step 3.

Step 3: Since $\mathcal{Q}$ is not complete, there exists a terminal element (s, i) of $\mathcal{Q}$ with $i > 0$. Let $s' = (s_1, \ldots, 0_i, \ldots, s_n)$ and $s'' = (s_1, \ldots, 1_i, \ldots, s_n)$. Augment $\mathcal{T}$ by removing $\mathcal{Q}$ and replacing $\mathcal{Q} \cup \{(s', h'), (s'', h'')\}$ for each pair (h', h'') such that for some policy π, $\pi(s') = h'$ and $\pi(s'') = h''$. The possible values for h' will be either $\{0\}$ (if s' is a state in which no more testing is necessary) or $\{j \mid s'_j = -1\}$ (any Y_j not already tested); possible values for h'' are analogous with respect to s''. For any such pair (h', h''), set

$$L(\mathcal{Q} \cup \{(s', h'), (s'', h'')\}) = L(\mathcal{Q}) + \bar{E}_{h'}(s') + \bar{E}_{h''}(s'')$$

and

$$U(\mathcal{Q} \cup \{(s', h'), (s'', h'')\}) = U(\mathcal{Q}) + \bar{E}_i(s) - \bar{E}_{h'}(s') - \bar{E}_{h''}(s'')$$

$$- 1(h' = 0)P(\hat{s}') \sum_{j \ni s'_j = -1} t_j - 1(h'' = 0)P(\hat{s}'') \sum_{j \ni s''_j = -1} t_j .$$

Recompute the upper bound U by $U = \text{Min } U(\mathcal{Q}')$ over $\mathcal{Q}' \in \mathcal{T}$ and return to Step 2.

D. An Example

We conclude this chapter with a numerical example of the above branch and bound method. The origin of the example is a fault finding problem which will be fully described in Chapter 2.

We first remark that for any decision problem $(Y_1, \ldots, Y_n, A_1, \ldots, A_d)$, it is sufficient to know only the sets

$$\bar{A}_i = \left\{\omega \in \{0, 1\}^n \mid \omega \in A_i \ , \ P\{Y = \omega\} > 0\right\}.$$

All the quantities computed in the course of the algorithm remain unchanged when A_i is replaced by $\bar{A}_i$, and we could have assumed initially that the sets $A_1, \ldots, A_d$ only partition $\{\omega \in \{0, 1\}^n \mid P\{Y = \omega\} > 0\}$.

Data for example:

	$P\{Y=\omega\}$								
	0	1/74	0	0	7/74	3/74	42/74	21/74	
ω_1	1	1	1	0	1	0	0	0	$t_1 = 4$
ω_2	1	1	0	1	0	1	0	0	$t_2 = 3$
ω_3	1	0	1	1	0	0	1	0	$t_3 = 16$
	A_1				A_2	A_3	A_4	A_5	

Each column of the table represents an $\omega \in \{0, 1\}^3$. The sets $A_1, \ldots, A_5$ are singletons, so a policy must determine for each outcome of the random vector Y which elementary outcome $\omega \in \Omega$ has occured. Clearly, no policy can stop by making no tests. We recall that $\bar{E}_j(s) = t_j \sum_{\omega \in \Omega(s)} \bar{\Delta}_j(\omega)$ and

$$\bar{\Delta}_j(\omega) = \begin{cases} 1 & \text{if, for } \omega' = (\omega_1, \ldots, (1 - \omega_j), \ldots, \omega_n) \text{ , we have} \\ & P\{Y = \omega\} = 0 \text{ or } P\{Y = \omega'\} = 0 \text{ or } \omega \text{ and } \omega' \text{ belong to} \\ & \text{a common set among the sets } \{A_1, \ldots, A_d\} . \\ 0 & \text{otherwise.} \end{cases}$$

However, for our example, we can take

$$\bar{\Delta}_j(\omega) = \begin{cases} 1 & \text{if } \omega_j = 0 \text{ and } P\{Y = \omega\} > 0 \text{ and } P\{Y = (\omega_1, \ldots, 1_j, \ldots, \omega_n)\} = 0 \\ 0 & \text{otherwise} \end{cases}$$

without changing the value of $P\{Y = \omega\}\bar{\Delta}_j(\omega)$. This particular form for $\bar{\Delta}_j(\omega)$ always occurs for fault finding problems such as this one. For our example,

$\bar{E}_1(s^o) = 4(42/74) = 84/37$ corresponding to $\omega = (0, 0, 1)$

$\bar{E}_2(s^o) = 3(42/74) = 63/37$ corresponding to $\omega = (0, 0, 1)$

$\bar{E}_3(s^o) = 16(1/74 + 7/74 + 3/74) = 88/37$ corresponding to $\omega = (1, 1, 0)$, $\omega = (1, 0, 0)$ and $\omega = (0, 1, 0)$ respectively.

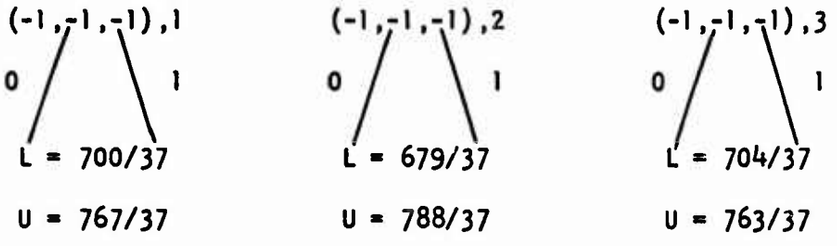
We have maintained a record of those $\omega \ni P\{Y = \omega\}\Delta_j(\omega) > 0$ for each j . When computing $\bar{E}_j(s)$ in steps to come, we can use this data rather than having to scan over the original data table. Following Step 1 of the algorithm, we will compute the upper and lower bounds associated with the trees $\mathcal{A}_i = \{(s^o, i)\}$, for $i = 1, 2, 3$.

$$L(\mathcal{A}_i) = \sum_{j=1}^{3} t_j - \sum_{j=1}^{3} \bar{E}_j(s^o) + \bar{E}_i(s^o)$$

$$= 23 - 235/37 + \bar{E}_i(s^o) = 616/37 + \bar{E}_i(s^o) .$$

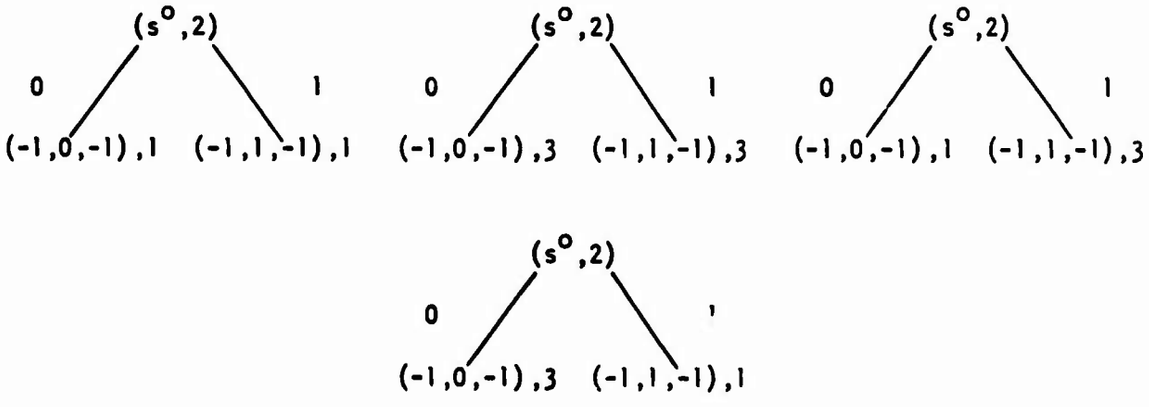
$$U(\mathcal{A}_i) = \sum_{j=1}^{3} t_j - \bar{E}_i(s^o) = 23 - \bar{E}_i(s^o) .$$

We will represent policy trees graphically, making it clear how they are a partial specification of policies. For example, the trees $\mathcal{A}_i = \{(s^o, i)\}$ and their bounds are:



The current upper bound is $U = \underset{i}{\text{Min}}\, U(\mathcal{A}_i) = 763/37$.

Proceeding to Step 2 of the algorithm, we see none of the three trees generated so far have lower bounds which exceed the current upper bound U . We see policy tree $\mathcal{A}_2$, representing all those policies which begin by testing Y_2 , has the lowest lower bound among all policy trees thus far generated. As this tree is not complete, we begin by extending this tree. A policy cannot stop in states $(-1,0,-1)$ or $(-1,1,-1)$ as knowledge of Y_2 alone is not sufficient to determine both Y_1 and Y_3 , regardless of the value Y_2 takes. Hence, $\{(s^o, 2)\}$ has four extensions:



To compute the appropriate lower bounds, we have

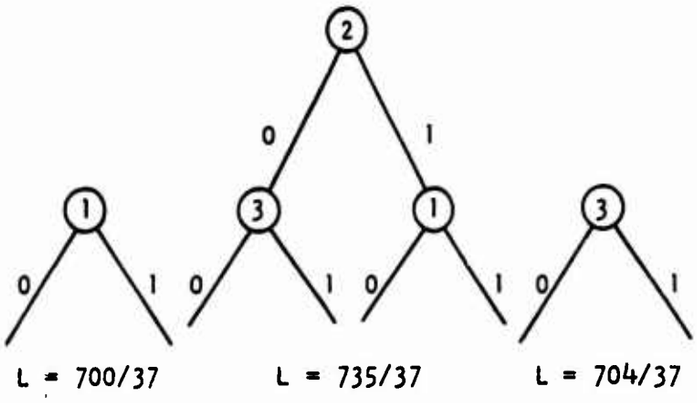
$\bar{E}_1(-1,0,-1) = 4(42/74) = 84/37$ corresponding to $\omega = (0, 0, 1)$

$\bar{E}_3(-1,0,-1) = 16(7/74) = 56/37$ corresponding to $\omega = (1, 0, 0)$

$\bar{E}_1(-1,1,-1) = 0$

$\bar{E}_3(-1,1,-1) = 16(1/74 + 3/74) = 32/37$ corresponding to $\omega = (1, 1, 0)$ and $(0, 1, 0)$ respectively.

We see that to compute say $\bar{E}_3(-1, 1, -1)$, we need only scan over $\omega \in \Omega(-1, 1, -1)$ (i.e., $\omega \ni \omega_2 = 1$) for which $P\{Y = \omega\}\Delta_j(\omega) > 0$. But this list of ω and corresponding $P\{Y = \omega\}$ has already been generated in computing $\bar{E}_3(s^o)$. The lower bounds for these newly created trees are easily computed. They are, respectively, 763/37, 767/37, 795/37, 735/37. The best (lowest) of the new upper bounds created is $U(a_2) + \bar{E}_2(s^o) - \bar{E}_1(-1, 0, -1) - \bar{E}_3(-1, 1, -1) = 735/37$, hence the new value of U is 735/37. Returning to Step 2, we see the first three of the four newly created trees can be dropped from further consideration, since their lower bounds exceed the current upper bound U on $T_{\pi_o}(s^o)$. We illustrate the remaining policy trees while suppressing the states involved.



Now, the first policy tree listed above has the lowest lower bound, and since it is not complete, we continue to Step 3. The computations are analogous to those made above.

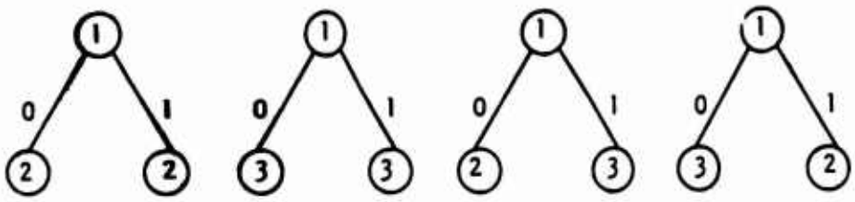
$\bar{E}_2(0,-1,-1) = 3(42/74) = 63/37$ corresponding to $\omega = (0, 0, 1)$

$\bar{E}_3(0,-1,-1) = 16(3/74) = 24/37$ corresponding to $\omega = (0, 1, 0)$

$\bar{E}_2(1,-1,-1) = 0$

$\bar{E}_3(1,-1,-1) = 16(1/74 + 7/74) = 64/37$ corresponding to $\omega = (1, 1, 0)$ and $(1, 0, 0)$ respectively.

The extensions of $\{(s^o, 1)\}$ are:



L = 763/37 L = 788/37 L = 827/37 L = 724/37

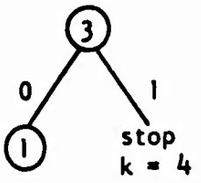
The best of the upper bounds generated is $U(\mathcal{Q}_i) + \bar{E}_1(s^o) - \bar{E}_2(0, -1, -1) - \bar{E}_3(1, -1, -1)$ or $724/37$, bringing the current value of U to $724/37$. Accordingly, the only policy trees remaining to be considered are:

L = 724/37 L = 704/37

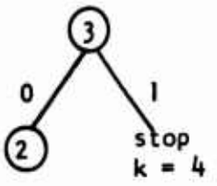
Expanding the second policy tree, we see if $Y_3 = 1$, then $Y_1 = Y_2 = 0$ a.s., so no more tests can be made. If $Y_3 = 0$, then a policy can test either Y_1 or Y_2. Comparing these two extensions (k indicates the "correct action", so $Y \in A_k$ a.s.):

$\bar{E}_1(-1,-1,0) = 0$

$\bar{E}_2(-1,-1,0) = 0$



L = 704/37



L = 704/37

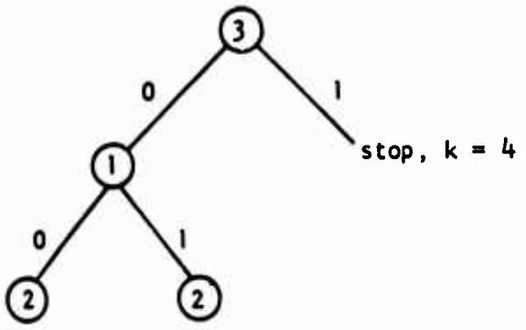
We see the upper bound generated by each policy tree is the same, namely

$$U(a_3) + \bar{E}_3(s^0) - P((-1,-1,1)) \sum_{j=1,2} t_j = 23 - 7(42/74) = 704/37 .$$

This improved value of U allows us to restrict our attention to the two trees just created above. We remark that our calculations have now fixed the value of 704/37 as the time of an optimal policy for this problem. Extending the first of these trees, we see only one extension is possible.

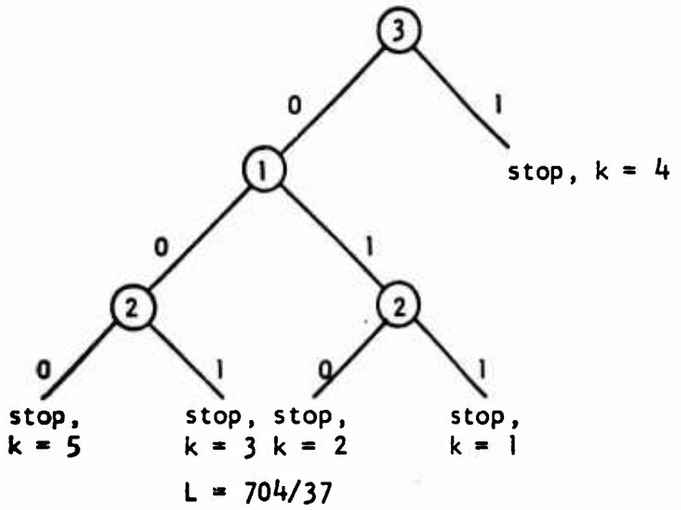
$\bar{E}_2(0,-1,0) = 0$

$\bar{E}_2(1,-1,0) = 0$



L = 704/37

There is no point in making any further upper bound computations. This newest tree still has the same lower bound, so we extend it again. There is only one extension, which gives a complete policy tree, at no increase in lower bound. Hence, we have



This tree is complete. It is a complete specification of a policy for the states relevant to that policy. It follows that for any π represented by this tree (they are all equivalent). $T_{\pi}(s^{o}) = 704/37$ and π is optimal.

E. Deadline Criterion

Another criterion with great intuitive appeal that might be applied to a policy is the deadline criterion. This is simply the probability of having completed testing, determining the correct action, on or before some specified deadline t. Policies in this case are rules telling the user which Y_j to test next, or to stop, depending on the current state of the problem and the time remaining until the deadline.

There is a branch and bound technique for solving this problem analogous to the one given above. We will describe a bound which could be used to generate an algorithm, however, it does not seem particularly well suited to the general problem. In Chapter 3 we will discuss special decision problem structures with the deadline criterion.

In order to generate a branch and bound procedure for this criterion, we need a lower bound on $P\{\tau_\pi > t \mid \hat{s}\}$ over all policies π, for each state s and time t for which $P(\hat{s}) > 0$, where $\tau_\pi(s)$ is the random time until the process stops, beginning in state s and following policy π. Equivalently, an upper bound on $P\{\tau_\pi(s) \leq t \mid \hat{s}\}$ over all policies π or an upper bound on $P\{\exists \text{ policy } \pi \ni \tau_\pi(s) \leq t \mid \hat{s}\}$ would be sufficient. The bound results by computing the probability of this event exactly. It would appear that to compute this probability each time the algorithm branches would make a considerably long computation. In particular, an incremental process by which lower bounds were able to be computed above no longer works here.

We will mention that the dynamic programming solution is easy to formulate, however, the optimal value function now depends on the state of the problem and the time remaining. This indicates that the functional equation must be solved about $3^n \times t$ times, where t is the initial deadline (assuming all times are integer).

CHAPTER 2

GENERAL FAULT FINDING MODELS FOR COHERENT SYSTEMS

Introduction

In this chapter we introduce coherent systems, a reliability model for "go, no-go' systems and give a description of some general fault finding problems which are solvable as decision problems. An example is included.

A. Coherent Systems

Coherent systems arise from the study of a physical system whose operation is classified as either functioning or failing, where this operation is determined by the joint functioning or failing of some finite set C of components. Accordingly, a joint performance of the components C is a function $X : C \to \{0, 1\}$ with the interpretation that for all $c \in C$,

$$X(c) = \begin{cases} 0 & \text{if component } c \text{ fails} \\ 1 & \text{if component } c \text{ functions .} \end{cases}$$

A system (not necessarily coherent) will be denoted by (C, φ), where φ is a function from all joint performances of the components to $\{0, 1\}$ with the interpretation that for all X,

$$\varphi(X) = \begin{cases} 0 & \text{if the system fails under joint performance } X \\ 1 & \text{if the system functions under joint performance } X . \end{cases}$$

Coherent systems are those systems for which the replacement of a failed component by a functioning one will not cause a functioning system to fail. To express this precisely, we say, for joint performances X and Y, $X \leq Y$ whenever for all $c \in C$, $X(c) \leq Y(c)$. Then coherent systems are those systems (C, φ) for which

(1) $$X \leq Y \Rightarrow \varphi(X) \leq \varphi(Y) .$$

We see some components may have no effect whatsoever on the system's performance. Such components are referred to as inessential components. Components not inessential are said to be essential. It follows that a component $c \in C$ is <u>essential</u> to (C , φ) if and only if

$$\exists\, X \ni \varphi(X , 0_c) \neq \varphi(X , 1_c) \text{ , where}$$

$$(X , K_c)(e) = \begin{cases} X(e) & \text{if } e \neq c \\ K & \text{if } e = c . \end{cases} \tag{2}$$

Some common examples of coherent systems are the series and parallel systems. A <u>series</u> system functions if and only if all its components function while a <u>parallel</u> system fails if and only if all its components fail. These are special cases of the k-out-of-n systems, (k/n) . A (k/n) <u>system</u> functions if and only if k or more of its n components function. In our notation, we say (C , φ) is a (k/n) <u>system</u> when C contains n components and $\varphi(X) = 1 \iff \sum_{c \in C} X(c) \geq k$.

Coherent systems have been studied in the monograph [Barlow and Proschan, 1956] as well as the literature. See [Birnbaum, Esary and Saunders, 1961] and [Esary and Proschan, 1963] for a basic exposition of coherent systems, particularly of the properties of the reliability function $P\{\varphi(X) = 1\}$, the probability the system functions, when the performance of each individual component is random and independent of the other components. An application of coherent systems to characterize a class of life distributions, those with increasing hazard rate average, is given in [Birnbaum, Esary and Marshall, 1966]. Coherent systems have also been studied under the guise of blocking systems. ([Butterworth, 1969] , [Fulkerson, 1968]).

Our definition of coherent systems in this chapter will differ from that found in [Birnbaum, Esary and Saunders, 1961] and elsewhere. A minor

difference is that our definition allows all components to be inessential. A significant difference is that while the assumption of independence, or in some instances the weaker assumption of association (e.g., see [Esary, Proschan and Walkup, 1966]), is usually made, we will allow the joint performance distribution to be arbitrary.

B. Some Fault Finding Models for Coherent Systems

Consider a coherent system (C , φ) for which the joint performance of the components is random according to a known joint distribution. Let the components be indexed $c_1 , \ldots , c_n$ and suppose one can test component c_i , requiring time t_i , to learn whether it is functioning or not. The object is to determine the state (function or failing) of each component a.s. by sequentially testing some components, in minimum expected time. A policy for testing is a feedback rule telling the user which component to test next or, if enough information has been gleaned, to stop, based on the previous tests and their results.

This is clearly a decision problem. To make the formal identification, let

$$Y_i = X(c_i) = \begin{cases} 0 & \text{if component } c_i \text{ is failed} \\ 1 & \text{if component } c_i \text{ is functioning .} \end{cases}$$

The sets $A_1 , \ldots , A_d$ can be taken as the singleton subsets of $\{0 , 1\}^n$, clearly a partition. Knowing the state of each component is then equivalent to knowing for which k we have $Y \in A_k$ a.s. We remark that for any decision problem, it is sufficient to know only the sets

$$A_i \cap \left\{ \omega \mid P\{Y = \omega\} > 0 \right\} ,$$

and we could have assumed initially that the sets $A_1 , \ldots , A_d$ partition $\left\{ \omega \mid P\{Y = \omega\} > 0 \right\}$.

We hasten to remark that the introduction of coherent systems in the above model serves only as a framework. The statement of the problem does not depend on the structure function φ of the coherent system (C , φ) . Indeed, the problem is completely specified by the distribution of the joint performance X and the testing times t_i .

The reason for this is the nature of our objective in testing the system. The object was to learn as much as possible about the system, namely the joint state of the components. This resulted in the partition $A_1 , \ldots, A_d$ of $\{0 , 1\}^n$ being the singleton subsets. We can formulate objectives for testing a set of components which require learning something less than everything from our testing, and which, as is to be expected, lead to a coarser partition of $\{0 , 1\}^n$ than the one above. Two such examples follow, in which it happens that the structure function φ does play a role in specifying the problem.

The first such model assumes, as above, that the joint performance X of a set of components C is random according to some known but arbitrary distribution. Suppose then that (C , φ) is a coherent system and that we wish to determine, in minimum expected time, whether the system is functioning or not when its components are performing according to X . As usual, we can test component c_i in time t_i to learn its state. This is a decision problem with $Y_i = X(c_i)$, $i = 1 , \ldots, n$, and with the partition $A_1 , \ldots, A_d$ having just two sets,

$$A_1 = \{\omega \mid \text{for } x(c_i) = \omega_i , i = 1 , \ldots, n ; \varphi(x) = 0\}$$

and its complement

$$A_2 = \{\omega \mid \text{for } x(c_i) = \omega_i , i = 1 , \ldots, n ; \varphi(x) = 0\}.$$

The framework of coherent systems provides a natural example of a problem in which our objective requires learning something less than everything about the system.

Another, somewhat artificial, example is what we will call the repair problem. As above, weare given a coherent system (C , φ) for which the components are performing according to the random joint performance X. In this case, a failed component can be repaired at a cost of $r_i \geq 0$ for component c_i. The object is to learn what is the least (repair) costly set of failed components, say R, which, when repaired, will insure the system's operation a.s. If one knew what joint performance $x : C \to \{0 , 1\}^n$ had occurred, it is clear that R would satisfy

$$\sum_{c_i \in R} r_i = \text{Min} \left\{ \sum_{c_i \in T} r_i \;\middle|\; T \text{ are failed components which, when repaired, insure the system's operation, i.e., for } y(c_i) = 1 \text{ if } x(c_i) = 1 \text{ or if } c_i \in T \text{, } y(c_i) = 0 \text{ otherwise, we have } \varphi(y) = 1 \right\}. \tag{3}$$

For example, the set R would be empty if no repairs were necessary, i.e. $\varphi(x) = 1$. For purposes of making this example well formed, assume that for every joint performance x with positive probability $(P\{X = x\} > 0)$, there is a unique set of R of failed components satisfying equation (3). This set R might be the same for several outcomes x, and consequently it might not be necessary to determine specifically the state of every component. In any case, we can ask for a testing policy which, in minimum expected time, determines just what the set of failed components R is. The resulting decision problem has $Y_i = X(c_i)$, $i = 1 , \ldots , n$, as usual. The partition $A_1 , \ldots , A_d$ is induced by the repair costs r_i and the structure function φ of the coherent system. Two elements ω and ω' of $\{0 , 1\}^n$ with positive probability will belong to a common set A_i whenever, for $x(c_i) = \omega_i$ and $x'(c_i) = \omega'$, $i = 1 , \ldots , n$, x and x' are joint performances for which the same set of components

R satisfy equation (3). Our assumption that R is unique for each x insures that this definition makes $A_1, \ldots, A_d$ a partition. If R is not uniquely determined by (3) for each joint performance x, the above problem formulation breaks down. Clearly one can, for each x, designate a specific R satisfying (3) and use it. However, this procedure can lead to a sub-optimal policy. Notice that we are content to restrict our attention to those ω for which $P\{Y = \omega\} > 0$.

We should explain that the following similar problem is different from the model described above. Given a testing time t_i and a repair time r_i for each component c_i, perform the necessary tests and repairs to the components in order to guarantee the system will operate a.s., all in minimum expected total time. Our solution of the repair problem would give a sub-optimal policy for this problem, since it would test enough components to determine the minimizing set if equation (3). However, it may really be optimal to stop testing prior to this. For example, if the sum of all the repair times for all components is less than every test time, any optimal policy would make no tests whatsoever and instead would make enough repairs so as to insure the system's operation a.s. on the basis of no tests, repairing all components if necessary. It appears that this problem can not be handled by the problem formulation of Chapter 1. It does, of course, have a straightforward dynamic programming solution.

C. An Example

Let us illustrate the models presented so far with an example. The coherent system for our example is one with three components, and is easiest to specify by the following sketch.

Figure 1

Connections through a box labelled c_i are present if and only if component c_i functions. The system functions if and only if there is a path through the network. Our system then functions whenever components c_3 and either c_1 or c_2 function. The distribution of the joint performance X is assumed to be the following: we suppose that nominally the components operate independently of each other, with $P\{c_1 \text{ functions}\} = 1/4$, $P\{c_2 \text{ functions}\} = 1/8$ and $P\{c_3 \text{ functions}\} = 2/3$. However, we suppose further that the system is down (not functioning), so the true distribution of X is the conditional distribution $P\{X = x \mid \varphi(X) = 0\}$. For the above system,

$P\{X = x\}$	0	1/74	0	0	7/74	3/74	42/74	21/74	Test Times
$x(c_1)$	1	1	1	0	1	0	0	0	$t_1 = 4$
$x(c_2)$	1	1	0	1	0	1	0	0	$t_2 = 3$
$x(c_3)$	1	0	1	1	0	0	1	0	$t_3 = 16$

Table showing the distribution of X and the testing times for our example

Figure 2

Each column of the table represents a joint performance x. The top row of the table shows the (conditional) probability that $X = x$ for each x. The table also shows the testing times for each component.

The first fault finding problem to consider, namely find the joint state of the components in minimum expected time, is the example in

Chapter 1. The calculations are carefully explained there for every step.

The second model described in this chapter, that is, determine with probability one whether the system will function or not when put in service, is simplified by the distribution of joint performance. Since the system will a.s. not function, no tests are necessary and because of the requirement that a policy must stop if possible, we see <u>every policy</u> is optimal as every policy must have $\pi(s^o) = 0$.

To specify the repair problem for this coherent system, we need to specify a repair cost for each component. Figure 3 below shows the set R of failed components realizing the minimum in equation 3 for each joint performance x with $P(X = x) > 0$. We have taken $r_1 = 3$, $r_2 = 2$, and $r_3 = 1$.

$R \rightarrow$	$\{c_3\}$	$\{c_3\}$	$\{c_3\}$	$\{c_2\}$	$\{c_3 , c_2\}$	
$x(c_1)$	1	1	0	0	0	$r_1 = 3$
$x(c_2)$	1	0	1	0	0	$r_2 = 2$
$x(c_3)$	0	0	0	1	0	$r_3 = 1$
	A_1	A_1	A_1	A_2	A_3	

Table showing the set R realizing the minimum in equation (3). Also shown are the sets A_1 , A_2 and A_3 which partition $\left\{\omega \mid P\{Y = \omega\} > 0\right\}$.

Figure 3

Notice the assumption that the set R realizing the minimum of equation (3) be unique is satisfied here. In Figure 3, we have also designated the sets A_1 , A_2 and A_3 . Of course, we identify each joint performance x with the $\omega \in \{0 , 1\}^3$ for which $\omega_i = x(c_i)$, $i = 1 , 2 , 3$.

To begin the algorithm for this example, we need to compute

$$\bar{E}_i(s^o) = t_i \sum_{\omega \in \Omega} P\{Y = \omega\}\, \bar{\Delta}_i(\omega) \quad \text{for} \quad i = 1, 2, 3, \text{ where}$$

$$\bar{\Delta}_i(\omega) = \begin{cases} 1 & \text{if, for } \omega' = (\omega_1, \ldots, (1 - \omega_i), \ldots, \omega_3), \text{ we have} \\ & P\{Y = \omega\} = 0 \text{ or } P\{Y = \omega'\} = 0 \text{ or } \omega \text{ and } \omega' \\ & \text{belong to the same } A_j,\ j = 1, 2, 3. \\ 0 & \text{otherwise.} \end{cases}$$

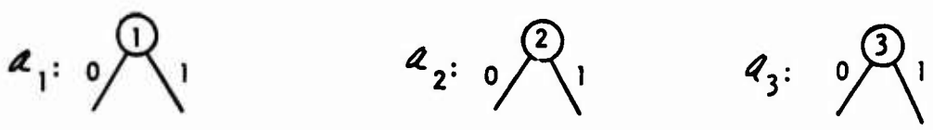
We have:

$$\bar{E}_1(s^o) = 4(1/74 + 3/74 + 42/74) = 92/37$$

$$\bar{E}_2(s^o) = 3(1/74 + 7/74 + 42/74) = 75/37$$

$$\bar{E}_3(s^o) = 16(1/74 + 7/74 + 3/74) = 88/37.$$

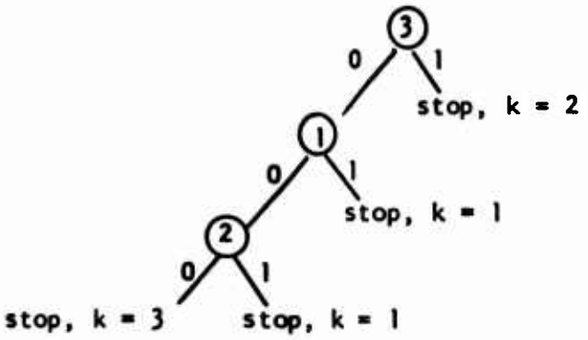
The practice of keeping a record of those ω for which $\bar{\Delta}_j(\omega)P\{Y = \omega\} > 0$ when $\bar{E}_j(s^o)$ is computed can be followed here, as well as for any decision problem. This practice tends to reduce some of the work necessary in computing $\bar{E}_j(s)$ for subsequent states s, at least in hand computations, and is detailed in the example of Chapter 1. Using equations (9) and (10) of Chapter 1, we begin with the following three policy trees and bounds.



$L(a_1) = 688/37$	$L(a_2) = 671/37$	$L(a_3) = 684/37$
$U(a_1) = 759/37$	$U(a_2) = 776/37$	$U(a_3) = 763/37$

Figure 4

The computations are fundamentally no different from those in the examples of Chapter 1, except of course for the choice of the sets $A_1, \ldots, A_d$. By following the branch and bound algorithm, the following optimal policy is generated.



Expected time to use this optimal policy is 692/37 . k is the correct actior· the random variable defined by the equation $Y \in A_k$ a.s.

Figure 5

D. Extensions to Problems with Modules

Within the study of coherent systems, the notion of a module has been studied. In [Birnbaum and Esary, 1965], a module is defined to be a subsystem of a coherent system. Specifically, a nonempty set $A \subset C$ is a module of (C , φ) if we can write:

$$\forall X , \varphi(X) = \psi(\Gamma(X|_A) , X|_{C-A}) ,$$

where

$X|_A$ is X restricted to the set A (hence a joint performance of the components A).

(A , Γ) is a coherent system, sometimes referred to as the module rather than A .

$(\{c_A\} \cup (C-A) , \psi)$ is a coherent system, with the pseudo-component c_A replacing the set A . The specification for the pseudo-component c_A is given by $\Gamma(X|_A)$.

For any coherent system, every singleton (one element) subset is a module. For the (k/n) systems other than the series and parallel, the only modules are the singletons. For the series or parallel systems, every nonempty proper subset of components is a module.

Our interest in modules will be to formulate a very general fault finding model. To extend one of the models previously mentioned, suppose that $M_1, \ldots, M_m$ is a modular partition of C for the coherent system (C, φ). By this we mean each M_i is a module of (C, φ) and $M_1, \ldots, M_m$ is a partition of C. If (M_i, Γ_i) is the coherent system for module M_i in the definition of a module, then our problem might be to determine the joint state of these modular subsystems in minimum expected time, by testing individual components. Let X denote the random joint performance of the components C for (C, φ). Then this problem is a decision problem, with $Y_i = X(c_i)$, $i = 1, \ldots, n$. There are 2^m A_j's, one for each joint state of the random vector $\Gamma_1\left(X|_{M_1}\right), \ldots, \Gamma_m\left(X|_{M_m}\right)$. This model reduces to a previous one when each M_i is a singleton.

A general fault finding model which relies on modules is one in which we suppose there are two sets of modules, the observable modules $Q_1, \ldots, Q_q$, and the repairable modules $R_1, \ldots, R_r$. The idea is that tests can be performed on any of the observable modules to determine whether they are functioning or not; we are limited to repairing or replacing only repairable modules. In order to make this problem well-formulated, assume that the joint state of the repairable modules is a.s. some function of the joint state of the observable modules. We can say then, knowing the joint state of the observable modules a.s. implies we know the joint state of the repairable modules. The decision problem is

to a.s. determine, in minimum expected time, the joint state of the repairable modules by testing the observable modules. Assume module Q_i requires time $t_i \geq 0$ to test, $i = 1, \ldots, q$. For this model, take $Y_i = \Gamma_{Q_i}\left(X|_{Q_i}\right)$, $i = 1, \ldots, q$ where $\left(Q_i, \Gamma_{Q_i}\right)$ is a typical observable module. There are 2^r A_j's, one for each joint state of the repairable modules.

E. Computational Experience

The branch and bound algorithm and the dynamic programming solution to the first fault finding model in Section B above were programmed on an IBM 360 (model 67) computer. Briefly, the problem is to determine the state of every component of a coherent system, in minimum expected time. Again we must remark that the problem is specified by the joint distribution of component performance; the coherent system only provides a framework in which to pose the problem.

The cases listed below are representative of those worked out. The distribution of component performance was nominal independence conditioned on the system being down (type 1) or conditioned on the system having just gone down (type 2). Thus, given a (k/n) system is down, there are at most $k - 1$ working components; given it just went down, there are exactly $k - 1$ working components. All joint distributions were generated by a (k/n) system, and a type (1 or 2) of conditional distribution. They are specified by k/n, type in Figure 6. These distributions are carefully detailed in the introduction to Chapter 3. Figure 6 gives a comparison of the two methods employed. The column "positive states" gives the number of joint performances x with positive probability, i.e. for which $P\{X = x\} > 0$. "Cycles" refer to the number of iterations the branch and bound method made. Some problems proved to be too large for

the branch and bound method to work in a reasonable length of time.

joint distribution	positive states	Computation time, in seconds: dynamic programming	branch and bound	cycles
2/3,1	4	.10	.04	3
3/3,1	7	.12	.08	5
2/4,1	5	.69	.20	7
2/4,2	4	.69	.34	11
3/4,1	11	.91	4.27	91
3/4,2	6	.85	2.30	49
4/4,1	15	1.04	54.65	319
4/4,2	4	.74	.39	11
2/5,1	6	4.17	1.46	21
2/5,2	5	4.17	2.65	36
4/5,1	26	7.16	not run	
4/5,2	10	5.97	> 600	
2/6,1	7	26.67	10.79	60
2/6,2	6	26.92	23.29	118
6/6,2	6	27.50	24.68	111

Comparison of Dynamic Programming versus Branch and Bound

Figure 6

The dynamic programming computation time grows roughly as 6^n, where n is the number of components. There is some variation in computation time for various joint distributions with the same number of components, however the number of positive states seems to give a rough indication of this variation, fewer positive states leading to shorter computation times.

Branch and bound computation times are seen to vary substantially, even for a fixed number of components. While in some cases this method was faster than dynamic programming by a factor of about 3, in other cases it was slower by several orders of magnitude. If a specific problem was being solved many times over for different input data, some

experience with both solution methods would be useful in determining which one handles that problem best.

One final observation to be made is that, for this specific problem, it seems that the largest problem either solution method could handle in, say less than 30 minutes, has eight or nine components. This may be a severe restriction on solving "real" problems.

CHAPTER 3

ANALYTICAL RESULTS FOR (k/n) SYSTEMS

Introduction

In this chapter we define several fault finding models and obtain some analytical results concerning them. The framework is that of coherent systems, introduced in Chapter 2. Chapter 3 is developed independently of Chapter 1.

A. Background

The basic problem comes about by considering a coherent system whose components are subject to being tested. The tests each require a known non-negative time to complete, and are performed sequentially until enough information has been learned. What constitutes enough information depends on the goal of the testing process.

Each model is specified by a coherent system (C, φ), a joint distribution for the set of performance indicators $\{X(c) \mid c \in C\}$, a goal for the testing process, either determining the state of every component, or determining the state of the system, and a criterion for comparing policies.

It is convenient to index the components in C, say $c_1, \ldots, c_n$, letting $Y_i = X(c_i)$ and $Y = (Y_1, \ldots, Y_n)$. Our choice of a coherent system will be limited to a k-out-of-n system, hereafter written (k/n). Recall that a (k/n) system is one of n components which functions if and only if k or more of its components function. In the above notation $\varphi(X) = 1 \iff \sum_{c \in C} X(c) \geq k \iff \sum_{i=1}^{n} Y_i \geq k$. In particular, $(1/n)$ systems are known as parallel systems and (n/n) as series systems.

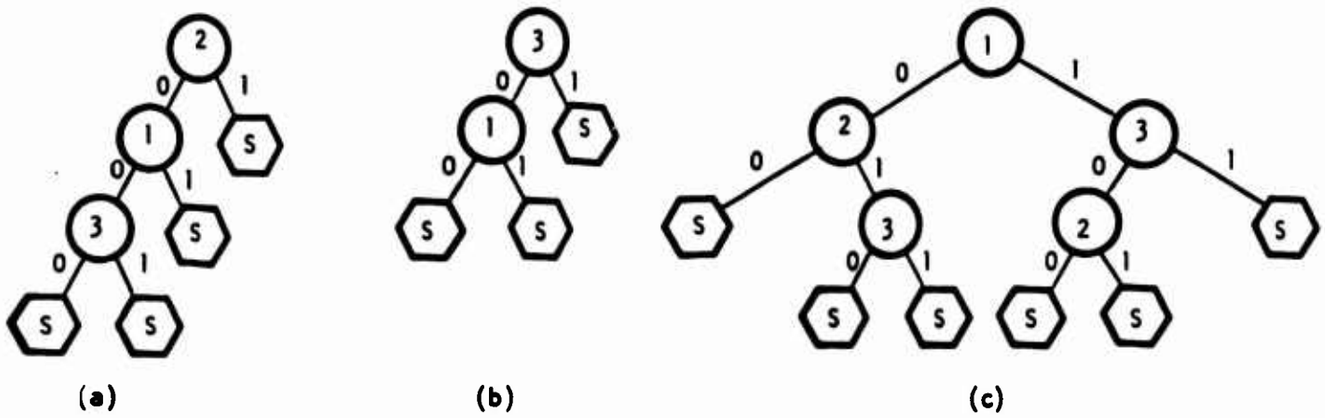
While independence is not always an appropriate distribution for the random variables $\{Y_1, \ldots, Y_n\}$, it does play a fundamental role in our

models. When we are concerned with determining the system's state, independence is assumed. When we seek the state of all the components, one of two distributions is assumed. The first is obtained by assuming that the components nominally operate independently, except that we know the system is down (not functioning). Hence the actual distribution is independence conditioned on the system being down. The second is again nominally independence. We suppose all components begin service in an operating state and function independently for some random lifetime, each with its own distribution. This process continues until the system fails, at which time every component is frozen at its current state. We suppose further that no two components have positive probability of simultaneous failure. We say the system has just gone down, that is, has at least one failed component which, when restored, will bring the system up again. For example, a series system which is down has one or more failed components while a series system that has just gone down has exactly one failed component.

The two criteria considered are mentioned in Chapter 1, the expected time to complete the testing process and the probability that the testing is completed on or before some deadline.

A policy is a rule which indicates which component should be tested next, based on the results of tests already made, provided any further testing is necessary. In Chapter 1 policies were functions, giving a component to test or an indication that testing could cease, for every possible state of our knowledge about the problem. While this formal representation was useful in Chapter 1, a more informal (but equivalent) representation is used here. Namely, a policy will be a certain kind of directed graph. Each node is either a stop node, meaning the testing process can cease upon reaching this point, or a test node, meaning more

testing is necessary. The stop nodes are labeled S and have no arcs leaving them. The test nodes are labeled i, $1 \leq i \leq n$, indicating component c_i is to be tested at this juncture, and have exactly two arcs leaving them. One arc is labeled 0, the other arc 1, corresponding to the event of finding component c_i failed or working respectively. Each node has exactly one arc entering it, except a single source node corresponding to the beginning of the testing process. Figure 1 illustrates typical policies for a (2/3) system with several choices of a goal and joint distribution. The forward orientation of each arc is downward.



(a) typical policy for finding the state of all components of a (2/3) system when the joint distribution of component performance is nominally independence, conditioned on the system being down.

(b) same as (a), except that the nominal distribution is conditioned on the system having just gone down.

(c) typical policy for finding the system's state for a (2/3) system when the joint distribution of component performance is independence.

Figure 1

The reader should be able to verify that each node labeled S really is a juncture at which testing can cease. It should be clear how these graphs are intended to be used as policies. The component to be tested initially is indicated in the source node, for example c_2 in policy (a) above. Upon testing a component and finding it, say failed, we proceed to the node connected to our present node by the arc labeled 0 and stop or make a test as indicated by the node at our present position. In order to avoid useless complications, it is assumed that a policy does not test the same component more than once. Of course, this does not prevent two nodes from indicating a test of the same component, for example policy (c) of Figure 1. These assumptions are also made in the policy definition of Chapter 1.

Policies (a) and (b) of Figure 1 illustrate that in some instances, a policy can be specified by an ordering of the components. We will refer to policies which are specified by orderings of the components as sequential policies. More precisely, a <u>sequential policy</u> is a policy π for which there is an ordering $a_1, \ldots, a_n$ with the property that π begins by testing the component labeled a_1, continuing sequentially to $a_2, \ldots$, etc., until enough information has been learned so testing can stop, or until all components have been tested, at which time testing can certainly cease. The order in which components are tested does not depend on test outcomes.

For some fault finding models, every policy is a sequential policy, (a) and (b) of Figure 1 for example, including of course any optimal policy. On the other hand, not all policies are sequential, as shown by policy (c) of Figure 1. An interesting feature of each fault finding problem solved in Chapter 3 is that each one has an optimal policy which is sequential.

B. Finding the State of All Components in Minimum Expected Time

The first specific models treated in this chapter have as their criterion the expected time to complete the testing process. This is the criterion for which the branch and bound algorithm of Chapter 1 is developed and is simply the expected value of the total testing time incurred. The total testing time is the sum of the times for each individual test made.

Consider first the problem of finding the state of every component of a (k/n) system, when the distribution of joint component performance is independence conditioned on the system being down. Recall that

$$(1)\quad Y_i = X(c_i) = \begin{cases} 0 & \text{if component } c_i \text{ is failed} \\ 1 & \text{if component } c_i \text{ is functioning,} \end{cases} \quad i = 1, \ldots, n .$$

Let

(2) t_i = time required to test component c_i ,

(3) p_i = probability (under nominal independence) that component c_i is functioning,

(3a) $q_i = 1 - p_i$, and let

(4) $P\{\cdot\}$ be nominal independence of the random variables $Y_1, \ldots, Y_n$.

Then we have $P\{Y_i = 1\} = p_i$, $i = 1, \ldots, n$. If (C, φ) is the (k/n) system, the distribution with respect to which we seek to minimize the expected testing time is $P\{\cdot \mid \varphi(X) = 0\}$, equivalently $P\{\cdot \mid \sum_{i=1}^{n} Y_i < k\}$.

The only assumption made about the parameters is

(5) $t_i \geq 0$ and

(6) $0 < p_i < 1$. $i = 1, \ldots, n$.

We will show below that there is an optimal policy for this problem which is sequential. Furthermore, the ordering to obtain this policy is straightforward; simply index the components so that $t_i q_i / p_i \leq t_{i+1} q_{i+1} / p_{i+1}$

$i=1,\ldots,n-1$, and test in the order $c_1,\ldots,c_n$ until $k-1$ working components are encountered, when testing can cease as any remaining components are almost surely failed, or until all components have been tested. This result is proven by induction on n, the number of components. The substance of this result is in the fact that, after testing a component, if it is failed we are faced with the situation of a $(k/n-1)$ system, or if it is functioning, the situation of a $(k-1/n-1)$ system, for the remaining $n-1$ components. To say this more precisely, let

(7) T_π = expected test time required of policy π, and if π begins by testing component c_i,

(8) $T_\pi(Y_i = \delta)$ = expected remaining testing time required of policy π after the initial test of Y_i, given $Y_i = \delta$, where $\delta = 0$ or 1.

Lemma 1: Let π be a policy for finding the state of all components in a (k/n) system, given the system is down. Let $1 < k < n$ and suppose policy π begins by testing c_i. Then

$$(9)\quad T_\pi = t_i + P\{Y_i = 0 \mid \sum_{j=1}^{n} Y_j < k\}\, T_\pi(Y_i = 0) + P\{Y_i = 1 \mid \sum_{j=1}^{n} Y_j < k\}\, T_\pi(Y_i = 1) .$$

For $\delta = 0$ or 1, the conditional distribution

$$P\{\cdot \mid \sum_{j=1}^{n} Y_j < k\ ,\ Y_i = \delta\}$$

is the same distribution on $\{Y_j \mid j \neq i\ ,\ j = 1,\ldots,n\}$ as

$$P\{\cdot \mid \sum_{\substack{j=1 \\ j\neq i}}^{n} Y_j < k - \delta\} ,$$

namely the one which results when components $\{c_j \mid j \neq i\ ,\ j = 1,\ldots,n\}$ form a $(k-\delta/n-1)$ system, given the system is down. In particular,

$T_\pi(Y_i = \delta)$ is the expected time to test this system, using as a policy $\pi(\delta)$, where $\pi(\delta)$ = the subgraph of π beginning with the node reached after c_i is tested and $Y_i = \delta$, and all nodes reachable from it.

Proof: Equation (9) follows by conditioning on the outcome of the first test π makes. To show the second assertion, let A be any event defined on the random variables $\{Y_j \mid j \neq i, j = 1, \ldots, n\}$. Then

$$P\{A \mid \sum_{j=1}^{n} Y_j < k, Y_i = \delta\} = P\{A \mid \sum_{\substack{j=1 \\ j\neq i}}^{n} Y_j < k, Y_i = \delta\} = P\{A \mid \sum_{\substack{j=1 \\ j\neq i}}^{n} Y_j < k - \delta\}$$

by independence of $Y_1, \ldots, Y_n$ under $P\{\cdot\}$. The remaining assertion says that $T_\pi(Y_i = \delta)$ can be thought of as the expected time to use $\pi(\delta)$, a policy, to test $\{c_j \mid j \neq i, j = 1, \ldots, n\}$ as if these components formed a $(k-\delta/n-1)$ system, given it is down. But this follows because the conditional distribution of performance indicators for the components $\{c_j \mid j \neq i, j = 1, \ldots, n\}$, given $\{Y_i = \delta\}$, is as shown above.//

Proposition 1:

Consider the problem of finding the state of all components of a (k/n) system in minimum expected time, given the system is down. If the components are indexed so that $t_i q_i/p_i \leq t_{i+1} q_{i+1}/p_{i+1}$, $i=1, \ldots, n$, then the sequential policy which tests $c_1, \ldots, c_n$ until k-1 working components are found or all components have been tested, is optimal.

Proof: We begin with showing the result for (1/n) and (n/n) systems directly. The case (k/n) is proven by induction on n.

For (1/n), or parallel systems, we see all components are a.s. failed, so no tests are necessary.

For (n/n) , or series systems, there is only one situation in which any policy can stop short of testing all components, that is when the first $n - 1$ tests each reveal a working component. In this instance, a policy can stop, since the one untested component is almost surely failed. In any other circumstance, all components must be tested. Hence, a policy π , when c_i is the one component which might not be tested by π , has

$$T_\pi = \sum_{j=1}^{n} t_j - t_i \, P\{Y_j = 1 \, , \, j \neq i \mid \sum_{j=1}^{n} Y_j < n\}$$

$$= \sum_{j=1}^{n} t_j - t_i \, (\prod_{j \neq i} p_j) q_i \, / \, (1 - \prod_{j=1}^{n} p_j) \quad .$$

Minimizing this quantity as i varies is equivalent to maximizing $t_i q_i / p_i$. For the sequential policy we propose as optimal, $i = n$. Because of the assumed ordering, it is clear this policy is optimal. In fact, any policy which does not test c_n in the eventuality that $c_1 \, , \ldots , \, c_{n-1}$ are working is optimal.

For (k/n) systems in general, we proceed by induction on n .

Basis: $(n = 2)$ See above for proof of result when $k = 1$ or 2.

Induction: $(n > 2)$ When $k = 1$ or $k = n$, appeal to the direct proof above. Suppose then that $1 < k < n$. We appeal to Lemma 1 above and the hypothesis of the induction. Among those policies (sequential or not) which begin by testing c_i , none is better than the sequential policy π_i given by $i, 1, 2, \ldots, i - 1 , i + 1, \ldots, n$. This is because for every policy π which begins by testing c_i , equation (9) holds; the quantities $T_\pi(Y_i = 0)$ and $T_\pi(Y_i = 1)$ are both minimized by the sequential policy π_i , due to the interpretation given them in Lemma 1 and the induction hypothesis. It now remains to show that it is optimal to test c_i first. Comparing the

sequential policy π_i' given by the sequence

$$1, i, 2, \ldots, i-1, i+1, \ldots, n,$$

and the sequential policy π_i, we see for $k > 2$, both policies have the same value because each must make at least two tests, that is, $T_{\pi_i'} = T_{\pi_i}$. The case of $k = 2$ is illustrated by Figure 2. Equations (10) and (11) are similar to equation (9).

π_i' π_i

Figure 2

$$(10)\quad T_{\pi_i'} = t_1 + t_i\, P\{Y_1 = 0 \mid \sum_{j=1}^{n} Y_j < 2\} +$$

$$P\{Y_i = 0,\ Y_1 = 0 \mid \sum_{j=1}^{n} Y_j < 2\}\, T_{\pi_i'}(Y_1 = 0,\ Y_i = 0)$$

$$(11)\quad T_{\pi_i} = t_i + t_1\, P\{Y_i = 0 \mid \sum_{j=1}^{n} Y_j < 2\} +$$

$$P\{Y_1 = 0,\ Y_i = 0 \mid \sum_{j=1}^{n} Y_j < 2\}\, T_{\pi_i}(Y_i = 0,\ Y_1 = 0)$$

The last terms in equation (10) and (11) are the same, since policies π_i and π_i' agree after two tests. Hence π_i' is better than π_i if

$$t_i \left(1 - P\{Y_i = 0 \mid \sum_{j=1}^{n} Y_j < 2\}\right) -$$

$$t_1 \left(1 - P\{Y_1 = 0 \mid \sum_{j=1}^{n} Y_j < 2\}\right) \geq 0$$

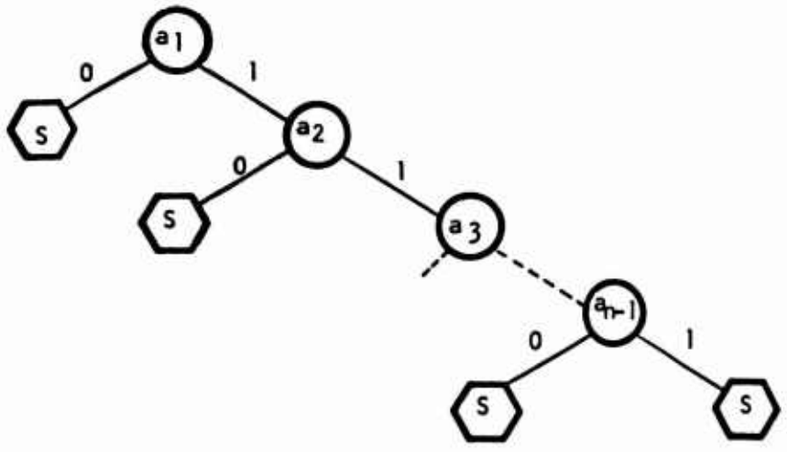
equivalently if $t_1 q_1/p_1 \leq t_i q_i/p_i$. Since this is clearly satisfied for each i , we conclude π_i' is as good as π_i , but then as we remarked above, the sequential policy π_1 given by $1, \ldots, n$ is the best policy which tests c_1 first. Hence $T_{\pi_1} \leq T_{\pi_i'} \leq T_{\pi_i}$, showing that of all policies which make at least one test, π_1 is as good as any. Since any policy must make at least one test, we conclude the sequential policy π_1 is optimal.//

This pleasing result does not continue to apply when we change the assumptions of the model slightly. Let's consider the problem which is identical to the one above, except that the distribution of the performance indicators is nominally independence conditioned on the system having just gone down. For (k/n) systems, this is just $P\{\cdot \mid \sum_{j=1}^{n} Y_j = k - 1\}$, where again the unconditional $P\{\cdot\}$ is independence on the random variables $\{Y_1, \ldots, Y_n\}$. Recall we wish to find the state of all components in minimum expected time.

We can obtain, in a special case, a similar result for (k/n) systems. The technique for proving these results is the same as for the results obtained above. For this reason, we choose to sketch the proofs, omitting the repetitious details. The notation and assumptions of equations (1) through (8) continue to apply.

Consider first the series system of n components, the (n/n) system. Since one component is failed and the remainder working, a.s.,

we see any testing policy stops as soon as a failed component is found, or n-1 components are found to be functioning. For this reason, every policy is in fact a sequential policy, as illustrated in Figure 3.



Typical policy for series system of n components, given system just went down. $a_1, \ldots, a_n$ is a permutation of $1, \ldots, n$.

Figure 3

When we speak of a policy π given by the sequence $a_1, \ldots, a_n$, we mean as shown in Figure 3. Lemma 1 above has its analogue here.

Lemma 2: Consider π, given by $1, \ldots, n$, as a policy for finding the state of all components of a (n/n) system, given the system has just gone down. Supposing $n \geq 2$, we have

$$(12) \quad T_\pi = t_1 + P\{Y_1 = 1 \mid \sum_{j=1}^{n} Y_j = n - 1\}\, T_\pi(Y_1 = 1) .$$

The conditional distribution $P\{\cdot \mid \sum_{j=1}^{n} Y_j = n - 1 , Y_1 = 1\}$ is the same distribution on $\{Y_2, \ldots, Y_n\}$ as $P\{\cdot \mid \sum_{j=2}^{n} Y_j = n - 2\}$. Hence

$T_\pi (Y_1 = 1)$ is the expected time to use policy $\pi(Y_1 = 1)$ defined by $2 ,..., n$, to test the components $\{c_2 ,..., c_n\}$ arranged in series, when this system has just gone down.

Proof: The method is the same as used in Lemma 1. Equation (12) results by conditioning on the results of the first test and realizing $T_\pi(Y_1 = 0) = 0$. The second assertion follows as easily. Letting A be any event on the random variables $\{Y_2 ,..., Y_n\}$, we have

$P\{A \mid \sum_{j=1}^{n} Y_j = n - 1 , Y_1 = 1\} = P\{A \mid \sum_{j=2}^{n} Y_j = n - 2\}$ by independence of of $Y_1 ,..., Y_n$, under $P\{\cdot\}$. This model has a simple rule for determining the optimal policy in a special case.

Proposition 2:

Consider the problem of finding the state of all components of a (n/n) system in minimum expected time, given the system just went down. Suppose $n \geq 2$ and also suppose there is an indexing of the components so that

(13) $t_i p_i / q_i \leq t_{i+1} p_{i+1} / q_{i+1}$, $i = 1 ,..., n$, and $t_{n-1} \leq t_n$

Then the policy given by the sequence $1 ,..., n$ is optimal.

Proof: The proof is by induction on n.

Basis: (n-2) Since any policy makes exactly one test, the optimal policy will test the component with shortest test time, namely c_1. (Notice we are only using $t_1 \leq t_2$.)

Induction: (n > 2) The argument proceeds as in proposition 1. Comparing policy π_i' given by $1, i, 2 ,..., i-1, i+1 ,..., n$, and policy π_i given by $i, 1, 2 ,..., i-1, i+1 ,..., n$,

we see $T_{\pi_i'} \leq T_{\pi_i}$ <=> $t_1p_1/q_1 \leq t_ip_i/q_i$. The policy π_1 satisfies $T_{\pi_1} \leq T_{\pi_i'}$, using lemma 2 and the hypothesis of of the induction. It follows π_1 is optimal, because $T_{\pi_1} \leq T_{\pi_i}$ and π_i is the best policy which begins by testing c_i .//

The (2/n) system has a similar result as just obtained for the series case. For the (2/n) system, the random variables $\{Y_1, \ldots, Y_n\}$ are nominally independent, conditioned on $\sum_{i=1}^{n} Y_i = 1$ or $Y_i = 1$ for one i . Every policy is again sequential, stopping as soon as a working component is found or n-1 components have been tested. Consider the random variables $\{Y_1', \ldots, Y_n'\}$ where $Y_i' = 1 - Y_i$. These random variables have the joint distribution of performance indicators for a series system which has just gone down. Furthermore, since knowing Y is equivalent to knowing Y' , every policy for the Y process is equivalent to a policy for the Y' process and conversely. Since proposition 2 indicates a special case for which the optimal policy is easily obtained for the Y' process, we are lead to conclude

Proposition 3:

Consider the problem of determining the state of every component in a (2/n) system which has just gone down. If the components are indexed so that

(14) $t_iq_i/p_i \leq t_{i+1}q_{i+1}/p_{i+1}$, $i = 1, \ldots, n-1$, and $t_{n-1} \leq t_n$.

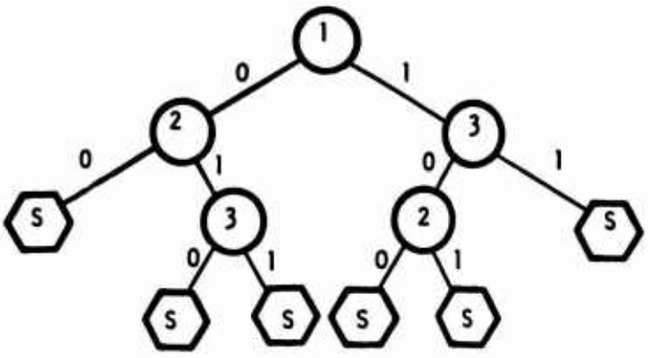
then the policy which tests in order of $c_1, \ldots, c_n$ until a working component is found, or c_{n-1} has been tested, is optimal.//

For a (k/n) system in general, not all policies are sequential, and indeed sometimes no sequential policy is optimal. Consider the

example illustrated by Figure 4

i	t_i	p_i	$t_i p_i / q_i$	$t_i q_i / p_i$
1	1	.9	9.	.111
2	10	.9	90.	1.11
3	10	.95	190.	.526
4	100	.95	1900.	5.26

Data table for (3/4) system



Policy π

Model is finding state of all components in minimum expected time, given system just went down. No sequential policy is optimal.

Figure 4

We will show that π is better than any sequential policy, and in fact, that π is optimal. Firstly, since the general form of π implies $T_\pi \leq 21$, π is better than any policy (sequential or not) which begins by testing c_4. Lemma 1 has its analogue for this model, and is proven just as lemma 1 is. Briefly, it says that if the first test finds a failed component, then the remaining components look like a (k/n-1)

system, while if the first component tested is working, the remaining components look like a (k-1/n-1) system. Using this, along with propositions 2 and 3, we see that among all policies, sequential or not, which begin by testing c_2, none is better than the sequential policy given by 2, 1, 3, 4, because

$$t_1p_1/q_1 \le t_3p_3/q_3 \le t_4p_4/q_4 \ , \ t_1q_1/p_1 \le t_3q_3/p_3 \le t_4q_4/p_4$$

and $t_3 \le t_4$. The same reasoning shows that the sequential policy given by 3, 1, 2, 4, is best among all policies which begin by testing c_3. However, the form of any sequential policy is such that the order of the first two tests can be reversed without changing the value of the policy. It follows that these two sequential policies use the same time as the sequential policies given by 1, 2, 3, 4, and 1, 3, 2, 4, respectively. Now, again using the reasoning above, the π illustrated in Figure 4 is as good as any which begins by testing c_1. Let π' and π'' be the sequential policies given by 1, 2, 3, 4, and 1, 3, 2, 4. It will follow that π is optimal and that no sequential policy is optimal, if $T_\pi < T_{\pi'}$ and $T_\pi < T_{\pi''}$. The following calculations are straightforward.

$$T_\pi = 11 + 10\,(1 - q_1q_2\,p_3p_4 - p_1q_2\,p_3q_4) \,/\, P\{\sum_{i=1}^{4} Y_i = 2\}$$

$$T_{\pi'} = 11 + 10\,(1 - q_1q_2\,p_3p_4 - p_1p_2\,q_3q_4) \,/\, P\{\sum_{i=1}^{4} Y_i = 2\}$$

$$T_{\pi''} = 11 + 10\,(1 - q_1p_2\,q_3p_4 - p_1q_2\,p_3q_4) \,/\, P\{\sum_{i=1}^{4} Y_i = 2\}$$

$$T_\pi < T_{\pi'} \iff p_3q_3 > p_2/q_2$$

$$T_\pi < T_{\pi''} \iff p_3q_3 > p_2/q_2$$

There is a special case, motivated by previous results, in which a sequential policy is easily obtained and is optimal.

Proposition 4:

Consider the problem of determining the state of every component in a (k/n) system which has just gone down. If the components are indexed so that

(15) $t_i p_i/q_i \le t_{i+1}p_{i+1}/q_{i+1}$, $t_i q_i/p_i \le t_{i+1}q_{i+1}/p_{i+1}$, $i = 1, \ldots, n-1$,

$t_{n-1} \le t_n$,

then the sequential policy given by the ordering $1, \ldots, n$ is optimal. (That is, test in the order $c_1, \ldots, c_n$ until $k-1$ working or $n-k+1$ failed components are found, at which time testing can stop.)

Proof: The method follows lemma 1 and proposition 1. Analogous to lemma 1, we see upon making the first test, the remaining $n-1$ components appear to form a $(k/n-1)$ or $(k-1/n-1)$ system, according as the first test finds a failed or working component, when $2 < k < n$. The argument proceeds exactly as proposition 1, by induction on n . The boundary case of $(1/n)$ is simple as no tests are necessary; all components are a.s. failed. The result for $(2/n)$ and (n/n) systems is contained in propositions 3 and 2 respectively. To complete the induction step, we only need to show $T_{\pi_i'} \le T_{\pi_i}$ where π_i' is the sequential policy given by the ordering $1, i, 2, \ldots, i-1, i+1, \ldots, n$, and π_i is given by $i, 1, 2, \ldots, i-1, i+1, \ldots, n$. But when $2 < k < n$, at least two tests are required of any policy, so $T_{\pi_i'} = T_{\pi_i}$ .//

C. Models Seeking the System's State in Minimum Expected Time

For the problem of determining the state of a (k/n) system, it is

assumed that the components operate independently. Clearly it would not be reasonable to condition on the system being down or having just gone down, since then no tests would ever be made. Notice also that independence is equally inappropriate when seeking the state of every component, because such a distribution would cause every policy to test every component. In this section, an optimal policy minimizes the expected testing time used. The notation and assumptions of equations (1) through (8) continue to apply.

The form of the results and the method of proof are analogous to the results obtained in the previous section. For the $(1/n)$, or parallel system, every policy is sequential and an optimal order in which to test is $c_1, \ldots, c_n$ whenever $t_i/q_i \le t_{i+1}/q_{i+1}$, $i = 1, \ldots, n-1$. For the (n/n), or series system, again every policy is sequential and an optimal order in which to test is $c_1, \ldots, c_n$ whenever $t_i/p_i \le t_{i+1}/p_{i+1}$, $i = 1, \ldots, n-1$. Similar to the last model of the previous section, for the (k/n) systems in general, sometimes no sequential policy is optimal. If the indexing can be performed so that both $t_i/q_i \le t_{i+1}/q_{i+1}$ and $t_i/p_i \le t_{i+1}/p_{i+1}$, $i = 1, \ldots, n-1$, then the sequential policy which tests in the order $c_1, \ldots, c_n$ stopping as soon as k working or $n-k+1$ failed components are found, is optimal.

D. Models with a Deadline Criterion

This section deals with a criterion mentioned only briefly in Chapter 1, the deadline criterion. To work with this criterion, let

(16) τ_π = random time to complete testing, using policy π.

Relating this notation to previous notation, we have

(17) $$T_\pi = E(\tau_\pi) .$$

The deadline criterion has a non-negative parameter t, the deadline; an optimal policy seeks to maximize the probability that $\tau_\pi \le t$.

The primary tool of the following results is contained in [Kadane, 1968] as his theorem 1, a variant of the Neyman-Pearson lemma. This result is

Theorem 1:

Let $\{p_i\}$ and $\{t_i\}$ be arbitrary non-negative sequences such that $\sum_i p_i < \infty$. Let X be the class of sequences $\{x_i\}$ such that $0 \le x_i \le 1$ for all i and let $b = \sum_{i \ni p_i > 0} t_i$. If $0 < t < b$, then the maximum of

$$\sum_i x_i p_i \tag{18}$$

subject to

$$\sum x_i t_i \le t \tag{19}$$

and $\{x_i\} \in X$ is attained, and it occurs when and only when

$$x_i = \begin{cases} 1 & \text{if } p_i > r\, t_i \\ 0 & \text{if } p_i < r\, t_i \end{cases} \tag{20}$$

for some r, $0 < r < \infty$, and

$$\sum_i x_i t_i = t \tag{21}$$

The set of r's satisfying (20) is the same for each optimal $\{x_i\}$ and is a single point or a closed interval.//

Kadane uses this result in his treatment of the ball-in-the-box problem with a budget ceiling. Briefly, this problem consists of searching a set of boxes for a hidden ball, while admitting the possibility of a search overlooking the ball. An optimal policy maximes the probability of finding the ball without exceeding a budget ceiling.

The first model, tested in detail, will serve to illustrate how theorem 1 is used to obtain analogous results for the remaining models. The problem is to find the state of all components of a (2/n) system for which the distribution of joint performance is nominal independence conditioned on the system being down. An optimal policy seeks to maximize the probability of having completed the testing process on or before the deadline t. Notice every policy is a sequential policy. Components are tested in some prearranged order until a working component is found, or all components have been tested.

The assumptions and notational conventions of equations (1) through (6) will continue to apply here. In addition, we will require that the deadline t be small enough to affect us, namely that

$$\sum_{i=1}^{n} t_i > t \tag{22}$$

Suppose policy π is given by the sequence $1, \ldots, n$. Then, for k defined by

$$0 \leq t - \sum_{i=1}^{k} t_i < t_{k+1} \tag{23}$$

we have

$$P\{\tau_\pi \leq t \mid \sum_{i=1}^{n} Y_i = 1\} = P\{Y_j = 1 \text{ for some } i = 1, \ldots, k \mid \sum_{i=1}^{n} Y_i \leq 1\} \tag{24}$$

$$= \left(\sum_{j=1}^{k} p_j \left(\prod_{i \neq j} q_i \right) \right) \Big/ \left(\sum_{j=1}^{n} p_j \left(\prod_{i \neq j} q_i \right) + \prod_{i=1}^{n} q_i \right) .$$

We see the problem is equivalent to finding an indexing of the components so that

$$\left(\sum_{j=1}^{n} p_j \left(\prod_{i \neq j} q_i \right) \right) \Big/ \left(\sum_{j=1}^{n} p_j \left(\prod_{i \neq j} q_i \right) + \prod_{i=1}^{n} q_i \right)$$

is a maximum, subject to $\sum_{j=1}^{k} t_j \leq t$. While theorem 1 does not address

itself to this problem exactly, the situations are similar. We will use theorem 1 in two ways to make statements about our problem. We remark first that, in using theorem 1, the finiteness of our model is no restriction. Also, there is always an optimal solution $\{x_i\}_{i=1}^{\infty}$ to the problem posed in theorem 1 with $0 < x_i < 1$ for at most one i .

The first application of theorem 1 provides a policy with a bound on the error incurred in using it rather than an optimal policy. Specifically,

Proposition 5:

Consider the problem of finding the state of every component of a (2/n) system, given the system is down, prior to the deadline t . Index the components so that

$$t_i q_i / p_i \leq t_{i+1} q_{i+1} / p_{i+1} \qquad i = 1, \ldots, n-1 \tag{25}$$

and let k be defined as in equation (23) above. Then, for the policy given by the sequence $1, \ldots, n$, equation (24) holds, and further, if π' is any optimal policy, we have

$$0 \leq P\{\tau_{\pi'} \leq t \mid \sum_{i=1}^{n} Y_i \geq 1\} - P\{\tau_{\pi} \leq t \mid \sum_{i=1}^{n} Y_i \leq 1\} \tag{26}$$

$$\leq f \cdot p_{k+1} \left(\prod_{i \neq k+1} q_i \right) / c$$

where

$$f = (t - \sum_{i=1}^{k} t_i) / t_{k+1} \quad \text{and} \quad c = \sum_{j=1}^{n} p_j \prod_{i \neq j} q_i + \prod_{i=1}^{n} q_i \tag{27}$$

In particular, if $f = 0$, equivalently if $\sum_{i=1}^{k} t_i = t$, then π is optimal.

Proof: Apply theorem 1, replacing p_i by $p_i\left(\prod_{j\neq i} q_j\right) / c$, $i = 1, \ldots, n$. Then for

$$x_i = \begin{cases} 1 & i < k+1 \\ f & i = k+1 \\ 0 & i > k+1 \end{cases} \tag{28}$$

$\{x_i\}_{i=1}^n$ is an optimal solution. A value of r generating this solution is $\left[t_{k+1} / \left(p_{k+1}\left(\prod_{i\neq k+1} q_i\right) / c\right)\right]^{-1}$. Equation (26) now follows by realizing that an optimal policy to the fault finding problem also provides a solution, although not necessarily optimal, to the problem posed by theorem 1. The value of the optimal solution is

$$P\{\tau_\pi \leq t \mid \sum_{i=1}^n Y_i \leq 1\} + \text{right hand side of (26).//}$$

We remark that the fault finding problem for this model is well known as the knapsack problem and as such, there are techniques available for solving it. Proposition 5 merely provides quick access to a sub-optimal policy and gives a bound on how much better any optimal policy might be. The form of this bound suggests how the model might be expanded to allow a partial test while making suitable assumptions about the usefulness of such a test, and how theorem 1 would be used to obtain an optimal solution in this case.

In order to obtain such a model to which theorem 1 would apply, assume one test can be conducted for any length of time up to the nominal time necessary for a complete test. Also assume, for $0 \leq f \leq 1$, and $i = 1, \ldots, n$,

(29) $P\{$a partial test of c_i, lasting ft_i, determines the state of $c_i\} = f$,

and the indicator of the event mentioned in equation (29) is independent of the joint state of the components, $(Y_1, \ldots, Y_n)$. Now a policy must

also indicate which one if any, of its tests is to be a partial test, and how long this test is to last. Because any tests concluded after the deadline t are irrelevant, as is the order in which tests concluded prior to t are performed, there is no loss of generality in requiring the partial test to begin just after the last complete test prior to t, and to end on or before t. Let π be the policy given by $a_1, \ldots, a_n$, with d the duration of the partial test. Suppose there is only time for at most k complete tests prior to t, equivalently that

$$0 \leq t - \sum_{i=1}^{k} t_{a_i} < t_{a_{k+1}} .$$

Then, if $d > 0$, the partial test is on $c_{a_{k+1}}$ and $\sum_{i=1}^{k} t_{a_i} + d \leq t$. Further, for $f = d/t_{a_{k+1}}$ we have when $k < n - 1$,

(30) $P\{\tau_\pi \leq t \mid \sum_{i=1}^{n} Y_i \leq 1\} = P\{$a working component is detected prior to $t \mid \sum_{i=1}^{n} Y_i \leq 1\} = \sum_{i=1}^{k} p_{a_i} \left(\prod_{j \neq i} q_{a_j} \right) / c + f\, p_{a_{k+1}} \prod_{j \neq k+1} q_j / c$,

and when $k = n - 1$,

(31) $P\{\tau_\pi \leq t \mid \sum_{i=1}^{n} Y_i \leq 1\} = P\{$a working component is detected or all components are found failed prior to $t \mid \sum_{i=1}^{n} Y_i \leq 1\}$

$$= \sum_{i=1}^{n-1} p_{a_i} \left(\prod_{j \neq i} q_{a_j} \right) / c + f \left(p_{a_n} \prod_{j \neq n} q_j + \prod_{j=1}^{n} q_j \right) / c ,$$

where the normalizing constant $c = \sum_{i=1}^{n} p_i \prod_{j \neq i} q_j + \prod_{j=1}^{n} q_j$ as above. The reason for the extra term in equation (31) is because when $k = n - 1$, a successful partial test will tell if not already known, the state of all components. This case is easily handled directly, since (31) shows the

probability that a policy is successful depends only on the last index a_n ; n-1 comparisons determine the optimal choice. When $k < n - 1$, it is clear theorem 1 applies directly, giving

Proposition 6:

For the problem of finding the state of every component of a (2/n) system, given the system is down, prior to the deadline t , when a partial test is allowed, index the components as in (25), a let k be defined by (23). If $k < n - 1$, then for π given by $1, \ldots, n$, with a partial test of c_{k+1} lasting $t - \sum_{i=1}^{k} t_i$, π is optimal.//

Let's consider the (2/n) system when the distribution of component performance is independence conditioned on the system having just gone down. Again every policy is sequential. Further, every policy makes at most $n - 1$ tests before stopping, so to have the deadline effective, we will assume

$$(32) \qquad \sum_{i=1}^{n} t_i - t_j > t \ , \quad j = 1, \ldots, n \ ,$$

in lieu of (22). Propositions 5 and 6 have their analogue here. The proofs are omitted.

Proposition 7:

Consider the problem of finding the state of every component of a (2/n) system, given the system just went down, prior to the deadline t . Index the components so that (25) holds, and let k be defined by (23). Then for π given by the sequence $1, \ldots, n$,

$$(33) \quad P\{\tau_\pi \leq t \mid \sum_{i=1}^{n} Y_i = 1\} = \sum_{i=1}^{k} p_i \left(\prod_{j \neq i} q_j \right) \Big/ \sum_{i=1}^{n} p_i \prod_{j \neq i} q_j \ ,$$

and if π' is any optimal policy, we have

$$(34)\quad 0 \le P\{\tau_{\pi'} \le t \mid \sum_{i=1}^{n} Y_i = 1\} - P\{\tau_{\pi} \le t \mid \sum_{i=1}^{n} Y_i = 1\}$$

$$\le f\, p_{k+1} \left(\prod_{j \ne k+1} q_j \right) \Big/ \sum_{i=1}^{n} p_i \prod_{j \ne i} q_j$$

where $f = \left(t - \sum_{i=1}^{k} t_i \right) / t_{k+1}$. In particular, if $\sum_{i=1}^{k} t_i = t$, then π is optimal.//

Similarly we have

Proposition 8:

For the problem of finding the state of every component of a (2/n) system, given the system is down, prior to the deadline t , when a partial test is allowed, index the components as in (25), and let k be defined by (23). If $k < n - 2$, then for π given by $1, \ldots, n$, with a partial test of c_{k+1} lasting $t - \sum_{i=1}^{k} t_i$, π is optimal.//

We will treat the series, that is (n/n) system only briefly. Notice that if $Y_1, \ldots, Y_n$ have the joint distribution for a series system, given the system just went down, then for $Y_i' = 1 - Y_i$, $i = 1, \ldots, n$, the Y' variables have the joint distribution of a (2/n) system, given the system just went down. This is the same transformation used in Section B above for the same purpose. It allows us to conclude that for a (n/n) system, given the system just went down, Propositions 7 and 8 hold, with the interpretation that

$$(35)\qquad P\{Y_i = 1\} = q_i \quad \text{and} \quad p_i = 1 - q_i \ .$$

The problem of finding the system's state for series or parallel systems, when the components operate independently, is treated just as the above models. One application of theorem 1 yields a sub-optimal policy and a bound on how much better an optimal policy might be, while

the second application involves the concept of a partial test, and an optimal policy in this case.

Consider first the series system of n components. To find the system's state, one must test until a working component is located, or all components have been tested, so every policy is sequential. To make the deadline effective, it is assumed (22) holds, that is, there is not enough time to test all the components prior to t. Let policy π be given by $a_1, \ldots, a_n$, and let k denote the maximum number of tests π can complete prior to t. Then

$$(36) \qquad P\{\tau_\pi \leq t\} = 1 - \prod_{j=1}^{k} p_{a_j} \quad \text{and} \quad \sum_{i=1}^{k} t_{a_i} \leq t .$$

The problem of finding a π which maximizes (36) is seen to be equivalent to seeking an indexing of the components which maximizes

$$\sum_{i=1}^{k} - \mathrm{Ln}\,(p_i)$$

subject to $\sum_{i=1}^{k} t_i \leq t$. Theorem 1, used here just as in proposition 5, yields

Proposition 9

Consider the problem of finding the system's state for a series system of n independent components, prior to a deadline t. Index the components so that

$$(37) \quad t_i/-\mathrm{Ln}\,(p_i) \leq t_{i+1}/-\mathrm{Ln}(p_{i+1}) , \quad i = 1, \ldots, n-1 .$$

Let the policy π test in the order of $1, \ldots, n$, and let π' be any optimal policy. Then

$$(38) \quad 0 \leq P\{\tau_{\pi'} \leq t\} - P\{\tau_\pi \leq t\} \leq \left(\prod_{j=1}^{k} p_j \right) \left(1 - (p_{k+1})^f \right) ,$$

where

(39) $$0 \leq t - \sum_{i=1}^{k} t_i < t_{k+1}$$

and

(40) $$f = \left(t - \sum_{i=1}^{k} t_i \right) / t_{k+1} \quad .$$

In particular, if $f = 0$, then π is optimal.

The second use of theorem 1 involves the concept of a partial test, however here the assumptions concerning such a test differ from those made above. The assumption of (29) is replaced by, for $0 \leq f \leq 1$ and $i = 1, \ldots, n$,

(41) $P\{$a partial test of c_i, lasting t_i, determines the state of $c_i\} = (p_i)^{f-1}$,

and, as before, the indicator of the event mentioned in (41) is independent of the joint state of the components $(Y_1, \ldots, Y_n)$. It follows that, if $k < n - 1$, $P\{$testing $c_1, \ldots, c_k$, then a partial test of c_{k+1}, lasting ft_{k+1}, determines the system's state$\} = 1 - \left(\prod_{j=1}^{k} p_j \right) (p_{k+1})^f$.

Using the same technique as above, we have

Proposition 10:

For the problem of finding the system's state of a n component series system, with independent components, prior to the deadline t, when a partial test is allowed, index the components as in (37), and let k be defined by (39). If $k < n - 1$, then for π given by $1, \ldots, n$, with a partial test of c_{k+1} lasting $t - \sum_{i=1}^{k} t_i$, π is optimal.

The case of a parallel system is handled by the transformation $Y_i' = 1 - Y_i$ as was done above. The conclusion is that propositions 9 and 10 hold for parallel systems, provided p_i is interpreted as $P\{Y_i = 0\}$.

SUMMARY

Chapter 1 deals with a class of combinatorial problems called "decision problems." We are given n binary random variables and a function of the vector of random variables. The object is to determine the value of this function by successively testing the random variables to learn their value, and then to stop when enough information is known to infer almost surely the function's value. Each random variable requires some known time to test; an optimal testing policy determines the function's value in minimum expected time.

Two solutions are proposed, a branch and bound solution and a dynamic programming solution. The functional equation arising from dynamic programming requires 3^n evaluations. However, the usual bonus (in dynamic programming) of having solved the original problem for every possible initial state occurs here, and so we have really solved a bigger problem.

The branch and bound solution first partitions the set of all policies into classes. Then it computes a lower bound on the value of a policy over policies in each specific class. The class with the smallest lower bound is itself partitioned. The algorithm terminates when a class containing essentially one policy is chosen for partitioning, since for such classes, the lower bound is in fact equal to the value of that policy. Since the value of the policy chosen this way is a lower bound on the value of every other policy, it is optimal. For the solution presented, the lower bounds are computed by adding to bounds previously computed, reducing some of the computational complexity. Also, formulas are given for computing an upper bound on the value of an optimal policy.

Using this, some classes of policies can be excluded directly, whenever their lower bound exceeds the upper bound, reducing some of the storage used by the algorithm.

Another criterion, the probability that a policy completes its testing prior to some deadline, is also discussed briefly.

Chapter 2 introduces coherent systems and gives some fault finding problems for them which can be handled by the methods of Chapter 1, for example, finding the state (functioning or failing) of every component, or finding the state of the system. Both algorithms of Chapter 1 were programmed for a computer to solve the problem of finding the state of each component of the system. A comparison of the computation time used by both methods on some problems is included. It can be seen that neither method is uniformly better than the other one for every problem. While branch and bound seems to be largely unpredictable, average computation time for dynamic programming grew roughly as 6^n. Since the functional equation was evaluated 3^n times, it appears that the time to evaluate this equation was proportional to 2^n.

The final chapter contains a few of the models from Chapter 2 which could be treated analytically. The coherent system in every case is a (k/n) system, that being one which functions if and only if k or more of its n components function. The problems of determining the system's state when the components operate independently, and of determining the state of all components when we condition on the system being down or having just gone down, are discussed. When an optimal policy can be identified, it turns out to be a sequential policy, i.e., one for which the components are tested in a predetermined order until testing can cease.

BIBLIOGRAPHY

Barlow, Richard E. and Proschan, Frank, Mathematical Theory of Reliability, 1965, John Wiley & Sons, Inc.

Birnbaum, Z.W. and Esary, J.D., "Modules of Coherent Binary Systems", J. Soc. Indust. Appl. Math., Vol. 13, No. 2, June 1965, 444-462.

Birnbaum, Z.W., Esary, J.D. and Marshall, A.W., "A Stochastic Characterization of Wear-Out for Components and Systems", Annals of Math. Stat., Vol. 37, No. 4, August 1966, 816-825.

Birnbaum, Z.W., Esary, J.D. and Saunders, S.C., "Multi-Component Systems and Structures and Their Reliability", Technometrics, Vol. 3, No. 1, February 1961, 55-77.

Butterworth, R.W., "Modules of Coherent Systems and Their Relationship to Blocking Systems", Operations Research Center Report, University of California, Berkeley, May 1969.

Esary, J.D. and Proschan, F., "Coherent Structures of Non-Identical Components", Technometrics, Vol. 5, No. 2, May 1963, 191-209.

Esary, J.D., Proschan, F. and Walkup, "A Multivariate Notion of Association, With a Reliability Application", Boeing Scientific Research Laboratories Document D1-82-0567, October 1966.

Fulkerson, D.R., "Networks, Frames, Blocking Systems", Mathematics of the Decision Sciences, Lectures in Applied Mathematics, Vol. 11, Amer. Math. Soc. (1968).

Kadane, Joseph B., "Discrete Search and the Neyman-Pearson Lemma", Jour. Math. Anal. and Appl., Vol. 22, 1968, 156-171.

Reinwald, Lewis T. and Soland, Richard M., "Conversion of Limited-Entry Decision Tables to Optimal Computer Programs I: Minimum Average Processing Time", Jour. Assoc. Comp. Mach., Vol. 13, NO. 3, July 1966, 339-358.

Reinwald, Lewis T. and Soland, Richard M., "Conversion of Limited-Entry Decision Tables to Optimal Computer Programs II: Minimum Storage Requirement", Jour. Assoc. Comp. Mach., Vol. 14, No. 4, October 1967, 742-755.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) University of California, Berkeley	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

A BRANCH-AND-BOUND METHOD FOR FAULT FINDING

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Research Report

5. AUTHOR(S) (First name, middle initial, last name)

Richard W. Butterworth

6. REPORT DATE August 1969	7a. TOTAL NO. OF PAGES 65	7b. NO. OF REFS 11
8a. CONTRACT OR GRANT NO. GK-1684 b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) ORC 69-21 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES Also supported by the U. S. Army Research Office-Durham under Contract DA-31-124-ARO-D-331.	12. SPONSORING MILITARY ACTIVITY NSF-GK-1684-Shephard, The National Science Foundation Washington, D.C. 20550

13. ABSTRACT

SEE ABSTRACT.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Unclassified

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Branch and Bound						
Fault Finding						
Coherent Systems						